REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Apr 90	Technical

4. TITLE AND SUBTITLE

Null Values in Definite Programs

5. FUNDING NUMBERS

DAAL03-88-K-0087

6. AUTHOR(S)

Yuan Liu and Jack Minker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Maryland
College Park, Maryland 20742

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 25870.29-MA

11. SUPPLEMENTARY NOTES

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT

Null values are a special kind of incomplete information that appear in database applications. There are several kinds of null values. The one modeled in this paper are those that represent entities that are known to exist but whose exact values are only known to be in a finite subset of constants in a given domain. For example, if we know that Paul is the fraternal grandfather of John, then we know that there is someone who is the father of John and a child of Paul. If we further assume that there are only finitely many individuals, then this someone can be represented by the kind of null value mentioned here.

In this paper we incorporate these null values into definite programs by using a new kind of symbols called S-constants. We present model theoretic, proof theoretic and fixpoint semantics for such programs. In the above example, given the additional knowledge that Mike is the father of Joe, these semantics allow us to answer the question "Are John and Joe brothers?" by "Yes, if the (unknown) father of John is Mike."

The proposed semantics reduce to the usual semantics for definite programs (e.g. [10]) when there are no null values present.

14. SUBJECT TERMS

incomplete information, null values, definite program semantics

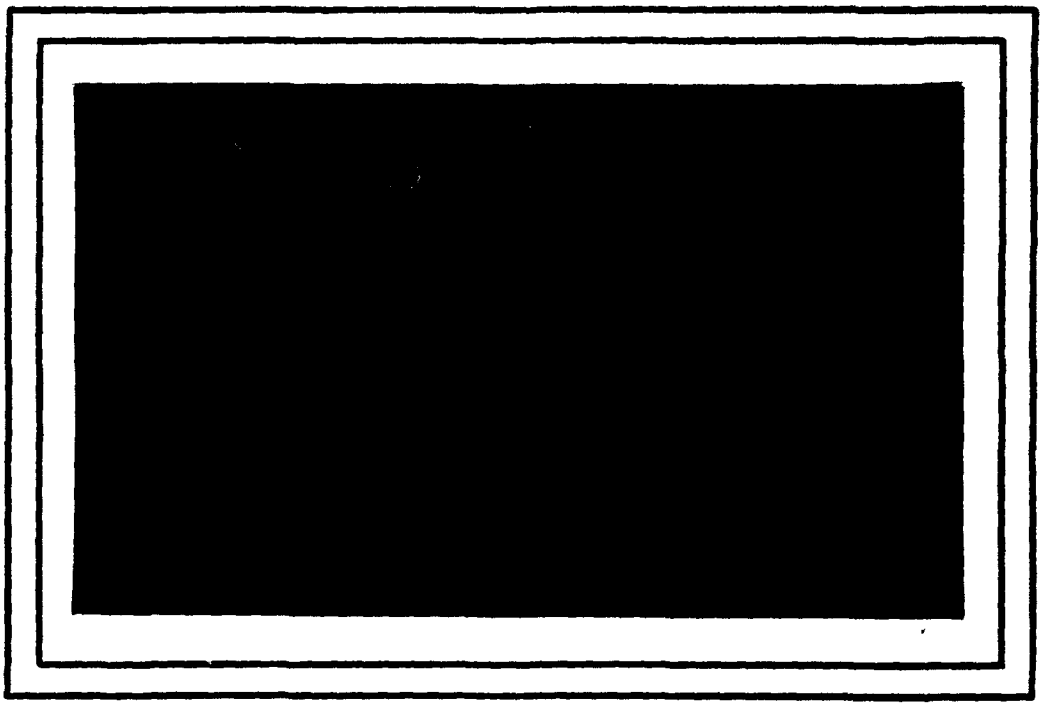
15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102



COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

UMIACS-TR-90-48
CS-TR-2447

April, 1990

Null Values in Definite Programs

Yuan Liu[1] and Jack Minker[1,2]

Department of Computer Science[1] and
Institute for Advanced Computer Studies [2]
University of Maryland
College Park, MD 20742

ABSTRACT

Null values are a special kind of incomplete information that appear in database applications. There are several kinds of null values. The one modeled in this paper are those that represent entities that are known to exist but whose exact values are only known to be in a finite subset of constants in a given domain. For example, if we know that Paul is the fraternal grandfather of John, then we know that there is someone who is the father of John and a child of Paul. If we further assume that there are only finitely many individuals, then this someone can be represented by the kind of null value mentioned here.

In this paper we incorporate these null values into definite programs by using a new kind of symbols called S-constants. We present model theoretic, proof theoretic and fixpoint semantics for such programs. In the above example, given the additional knowledge that Mike is the father of Joe, these semantics allow us to answer the question "Are John and Joe brothers?" by "Yes, if the (unknown) father of John is Mike."

The proposed semantics reduce to the usual semantics for definite programs (e.g. [10]) when there are no null values present.

Keywords: *incomplete information, null values, definite program semantics*

Table of Contents

Null Values in Definite Programs

Yuan Liu[1] and Jack Minker[1,2]

Department of Computer Science[1] and
Institute for Advanced Computer Studies [2]
University of Maryland
College Park, MD 20742

1. Introduction

A particular kind of incomplete information, called null values, often emerges in relational database applications. These represent the information "entity exists but value at present unknown". Consider the following example,

> Let database **DB** contain a single relation Father(F,S) where the domains of the attributes F and S are both male names. A tuple (f,s) is in the relation iff f is the father of s and s is a son of f. Assume that we have the tuple (John,Smith) in the database. We want to record the knowledge that Jim is the grandfather of Dave, i.e. someone is the father of Dave and a son of Jim, whose identity is unknown at this time. We also know that Joe has a son, but we do not know his name.

In this example, the unknown son of Jim, the father of Dave and the unknown son of Joe are null values.

These null values can be subdivided into two subtly different categories. The first contains those null values whose actual values can only be in a known domain. For instance, in the above example if we know that D is the set of all male names, then we know that Joe's son must be one of them, i.e. we can write '$\text{Father}(\text{Joe},\omega) \wedge \omega \in D$', where ω represents a null value.

This kind of null values are similar to what is termed OR-object in [8] which are used to denote a special class of disjunctive facts in relational databases. Lipski's incomplete information [9] is a generalization of such null values.

The second category are those null values whose actual values can be either in a set of known constants or can be entirely new constants. The null values in Reiter's work [12] fall into this category.

Most of the previous work on query answering in databases with null values, deal with extensional databases only, i.e. there are no deductive rules. Some of these try to extend the relational algebra operators to accommodate the appearance of null values, e.g. [1, 2, 3, 4, 5, 7, 13, 16]. Others use a proof-theoretic approach which treats databases as sets of first order theories and queries as theorems to be derived from these theories, e.g. [11, 12, 15, 6, 14].

In this paper, we consider the problem of adding to a definite program, incomplete information represented by the first kind of null values. One way of achieving this is to use disjunctive clauses. For example, in [6], the information 'the unknown son of Joe is one of Dave, Mike or Smith' is represented by the clause '$\text{Father}(\text{Joe},\text{Dave}) \vee \text{Father}(\text{Joe},\text{Mike}) \vee \text{Father}(\text{Joe},\text{Smith})$'. However, this means that we need to deal with the full complexity of disjunctive logic programs, although the clauses we use are just a very special subset of disjunctive clauses.

Instead, we preserve the definiteness of our clauses by introducing a new kind of symbol, called an S-constant, to represent these null values. This also allows us to define a refutation procedure that has the appearance of the SLD refutation. In the next section, we present background information. In the three following sections, we present, respectively, model semantics, fixpoint semantics, and procedural semantics for definite programs with null values. Finally, we discuss our results and future work.

1.1. Language

Two different ways have been described to model null values. One method uses a single symbol to represent any null value, e.g. [5]. In this method, whether or not two null values are equal is unknown. In this representation the database DB, given above, would have the following tuples in relation Father: (John,Smith), (Jim,ω), (ω,Dave) and (Joe,ω), where ω represents null values. The other methods use indexed null values (e.g. [14]) where different symbols are used to represent null values, and unlike the first method, whenever two null values are represented by the same symbol they are equal. In this representation the database DB, given above, would be represented as Father(John,Smith), Father(Jim,ω_1), Father(ω_1,Dave), and Father(Joe,ω_2). Whether or not two null values represented by different symbols, e.g. ω_1 and ω_2 above, are equal to each other is unknown.

The latter method is more expressive than the former because it can be used to record more information about null values, for instance that one unknown entity (the father of Dave) is the same as another (the unknown son of Jim), while the former method cannot. Also, the former can be viewed as a special case of the latter where every unknown value is represented by a distinct indexed null. Therefore, this latter approach is adopted here.

Our language is basically the same as that of first order logic (with equality symbol), except that we have a new kind of symbol called an *S-constant*, usually written with a lower case Greek letter. S-constants are used to represent null values. Other than the definition of terms, the definitions for constructs like atoms or formulas exactly parallel those for a first order language.

Definition D1.1 (***Formula***) A language L, contains symbols for the following: constants (e.g. a, b, c), S-constants (e.g. σ_1, σ_2), function symbols (e.g. f, g), variables (e.g. x, y, z), predicates (e.g. P, Q, R), connectives ($\neg$, $\vee$, $\wedge$, $\leftarrow$), quantifiers ($\forall$ and $\exists$), and punctuations ("(", ")", ",").

- A ***term*** is defined inductively as follows:
  - A variable is a term
  - A constant is a term
  - An S-constant is a term
  - If f is an n-ary function and $t_1,...,t_n$ are terms, then $f(t_1,...,t_n)$ is a term. If $n \geq 1$, then it is called a ***functional term***.
- A ***formula*** is defined inductively as follows:
  - If P is an n-ary predicate, and $t_1,...,t_n$ are terms, then $P(t_1,...,t_n)$ is a formula. $P(t_1,...,t_n)$ is also called an ***atom***.
  - F and G are formulas, then $\neg F$, $F \wedge G$, $F \vee G$, $F \leftarrow G$ are also formulas.
  - F is a formula and x is a variable, then $\forall xF$ and $\exists xF$ are also formulas.
- A formula (term) is ground if it contains no variable. A ***ground instance*** of an atom A is obtained by substituting each variable in A by a ground term. ■

In order to model null values whose actual values are in some known domains, we attach a specification of these domains to each formula in a program.

Definition D1.2 (***Program***) Let L be a language with constants D, and S-constants Σ.

- An ***extended program clause***, is a two-tuple (C;R). C is a formula of the form $A \leftarrow B_1,...,B_m$, where $m \geq 0$, A and B_j, $1 \leq j \leq m$, are atoms. R is a set of the form $\{\{\sigma_1\} \subseteq D_1,...,\{\sigma_n\} \subseteq D_n\}$ where for $1 \leq i \leq n$, σ_i is an S-constant that appears in C and D_i is a non-empty subset of D. We call 'A', '$B_1,...,B_m$' and 'R', respectively, the ***head***, the ***body***, and the ***domain specification*** of the extended program clause. D_i is called the ***range*** of σ_i in R. If there is no S-constant in a clause, then $R = \varnothing$.
- A ***program*** P is a set of extended program clauses that satisfy the following: if an S-constant, σ, appears in two different clauses, $(C_1;R_1)$ and $(C_2;R_2)$, then σ has the same range in R_1 and R_2. Let this range be r_σ. We say r_σ is the range of σ in program P. We will also call r_σ the ***initial range*** of σ. ■

Example 1.1
Let σ_1 and σ_2 be two S-constants and P be a program with the following extended program clauses:

(Father(σ_1,John);{$\{\sigma_1\} \subseteq$ {Mike,Smith,Joe}}) (E1.1.1)
(Father(Paul,σ_1);{$\{\sigma_1\} \subseteq$ {Mike,Smith,Joe}}) (E1.1.2)
(Father(σ_2,Jones);{$\{\sigma_2\} \subseteq$ {Dave,Mike,Smith}}) (E1.1.3)
(Father(Dave,Paul);∅) (E1.1.4)
(Sib(x,y) ← Father(z,x),Father(z,y);∅) (E1.1.5)

Note that σ_1 has the range {Mike,Smith,Joe}, and σ_2 the range {Dave,Mike,Smith}. This means that the possible values of σ_1 are Mike, Smith or Joe and those of σ_2 are Dave, Mike or Smith. ■

Informally, (E1.1.1) and (E1.1.2) together specify that someone, represented here by σ_1, is both the father of John and the son of Paul, and the domain specifications say that this "someone" can only be Mike, Smith or Joe. On the other hand, (E1.1.3) says that someone, represented by σ_2, is the father of Jones, and the domain specification restricts his identity to Dave, Mike or Smith. (E1.1.4) is an assertion that Dave is the father of Paul with no conditions associated with the statement.

There are two things to note. The first is that the "meaning" of σ_1 is determined by all the assertions that contain σ_1, i.e. (E1.1.1) and (E1.1.2). The second is that although σ_1 and σ_2 are two different S-constants, they may in fact be the same person since there are common elements, namely Mike and Smith, in their ranges. However, the information contained in the program is not sufficient to determine this equality one way or the other. We will formalize this discussion about the meaning of S-constants in later sections.

Next, we define the notion of a substitution as follows:

Definition D1.3 (***Substitution***) Let there be an arbitrary ordering of S-constants so that they are represented by σ_1, σ_2 ,.... etc, where $\sigma_i<\sigma_j$ iff i<j. A ***substitution*** θ is a finite set of the form $\{t_1/v_1,...,t_n/v_n\}$, where each v_i is either a variable or an S-constant, and $v_i \neq v_j$ for $i \neq j$. Each v_i does not appear in the corresponding t_i. Also, if v_i is a variable, then t_i is an arbitrary term, and if v_i is an S-constant, say σ_m, then t_i can only be either a (normal) constant or another S-constant σ_n such that $\sigma_m<\sigma_n$. The special substitution $\varepsilon=\{\}$, is called the ***identity substitution.***
The ***application*** of a substitution θ to a formula (or a term) H, written as $H\theta$, is the formula (term) obtained by simultaneously replacing all the v_i's in H by their corresponding t_i's. ■

The reason for imposing an arbitrary ordering on the S-constants will be made clear in the next section.

In the above definition, we do not allow substitutions of the form $\{t/\sigma\}$ where σ is an S-constant and t is a functional term other than a constant. This is because we consider function symbols to be distinct from constants, and since S-constants only have constants as their values, we do not consider S-constants to be replaceable by functional terms.

Composition of substitutions is defined in the usual way.

Definition D1.4 (***Composition of Substitutions***) Let $\theta = \{s_1/u_1,...,s_m/u_m\}$ and $\rho = \{t_1/v_1,...,t_n/v_n\}$ be substitutions. The ***composition***, $\theta\rho$, of θ and ρ is the substitution obtained from the set $\{s_1\rho/u_1,...,s_m\rho/u_m,t_1/v_1,...,t_n/v_n\}$ by deleting any item $s_i\rho/u_i$ for which $u_i=s_i\rho$ and deleting any item t_j/v_j for which $v_j \in \{u_1,...,u_m\}$. ■

2. Model Semantics

For logic programs without null values, the Herbrand Base of a program is an integral part of their declarative semantics. Therefore, in order to define a formal semantics for logic programs with null values we extend the classical Herbrand Base to include S-constants.

2.1. Herbrand Base with Null Values

First, we have the following straightforward extension to the Herbrand Base:

Definition D2.1 (***Null-Extended Herbrand Universe and Quasi Herbrand Base***) Let P be a program, and Σ be the set of S-constants in P.

- The Herbrand Universe with null values ${}_N U_P$ is the set of all ground terms that can be formed from constants, S-constants, and functions appearing in P. If no constant appears in P then we add an arbitrary constant, say a, to form ground terms.
- The *quasi*-Herbrand Base qHB(P) is the set
  $\{ P(c_1,...,c_n) \mid P$ is an n-ary predicate symbol in P ,c_i, i=1,...,n, are elements in ${}_N U_P\}$ ■

We note that when there is no S-constant, i.e. $\Sigma=\emptyset$, the quasi Herbrand Base reduces to the (classical) Herbrand Base.

The quasi Herbrand Base alone is not adequate for our purpose because it contains no information about the incompleteness of the S-constants. The following example illustrates what additional information we need by showing how we might want to respond to queries on a program containing such information. It exemplifies the kind of information we want to derive from a program.

Example 2.1
Consider the program in Example 1.1, which consists of the following:

(Father(σ_1,John) ; {{σ_1} ⊆ {Mike,Smith,Joe}})
(Father(Paul,σ_1) ; {{σ_1} ⊆ {Mike,Smith,Joe}})
(Father(σ_2,George) ; {{σ_2} ⊆ {Dave,Mike,Smith}})
(Father(Dave,Paul) ; ∅)
(Sib(x,y) ← Father(z,x),Father(z,y) ; ∅)

where σ_1 and σ_2 are S-constants.

In order for George and Paul to be siblings, σ_2 must be equal to Dave. This is possible since Dave is in the range of σ_2. On the other hand, for John and Paul to be siblings, σ_1 has to be Dave. However, this is in conflict with the information in the program since the range of σ_1 specifies that it can only be one of Mike, Smith or Joe. Now consider John and George. In order for them to be siblings, σ_1 and σ_2 must have the same value(s). This is possible since there are common elements, Mike and Smith, in their ranges. ■

The first two queries in the above example involve restricting the range of an S-constant to a single element; while the third one requires that we reconcile the values of two S-constants.

None of these queries can be answered with a definite "yes" or "no", even though none of them contains any S-constant. The restrictions we obtained, i.e. "σ_1 = Dave" or "$\sigma_1=\sigma_2$", must be recorded somehow so as to give a proper answer. That is, John and Paul are siblings if σ_1 is Dave, and John and George are siblings if σ_1 is the same as σ_2. Another reason for recording these restrictions which may not be apparent from the above example is that we may need them in further derivations.

Example 2.2
Let us add the extended program clause "(SR(x,y,u,v) ← Father(x,y),Sib(u,v) ; ∅)" to the program in the above example. Then in order for SR(Paul,Mike,John,George) to be true, Paul must be the father of Mike and John must be a sibling of George. As we see in the above example, John and George are siblings under the condition that $\sigma_1=\sigma_2$. On the other hand, for Paul to be Mike's father, σ_1 must be Mike (by (E2.1.2)). Therefore, the tuple (Paul,Mike,John,George) satisfies the relation SR if both of these conditions are true, i.e. $\sigma_1=\sigma_2$ and σ_1 = Mike.

Similarly, for the tuple (Paul,Joe,John,George) to be in relation SR, Paul must be the father of Joe and John must be a sibling of George. As before, for Sib(John,George) to be true, σ_1 must be equal to σ_2 while for Father(Paul,Joe) to be true, σ_1 must be Joe. Here, however, we see that the condition $\sigma_1=\sigma_2 \wedge \sigma_1$=Joe can not be satisfied, because $\sigma_1=\sigma_2$ implies that both σ_1 and σ_2 can only be either Mike or Smith, therefore, although Joe is in the original range of σ_1, the condition $\sigma_1=\sigma_2$ precludes it from the values σ_1 can take. ■

This example illustrates two points. First, we must record the conditions necessary to derive an answer, e.g. the condition $\sigma_1=\sigma_2$ for deriving Sib(John,George). In addition, we must also record how these conditions restrict the ranges of the S-constants involved in order to detect an unsatisfiable condition.

To achieve this, we generalize the domain specifications attached to program clauses such that to each ground atoms in the quasi-Herbrand base of a program, we attach a set of specifications of the form

$S \subseteq V$, where S is a set of S-constants, and V is a non-empty set of (normal) constants. $S \subseteq V$ represents the condition that all the S-constants in S are equal to one another, and V is the set of common values they can take.

For instance, in the example concerning SR(Paul,Mike,John,George), the set of specifications attached to the atom Father(Paul,Mike) would be $\{\{\sigma_1\} \subseteq \{Mike\}\}$, which means that σ_1 can only take the value Mike, and the specification attached to Sib(John,George) would be $\{\{\sigma_1,\sigma_2\} \subseteq \{Mike,Smith\}\}$, meaning that σ_1 and σ_2 must have the same values and these values can only be either Mike or Smith. The specification attached to SR(Paul,Mike,John,George) should be $\{\{\sigma_1,\sigma_2\} \subseteq \{Mike\}\}$. Similarly, the specification attached to Father(Paul,Joe) would be $\{\{\sigma_1\} \subseteq \{Joe\}\}$. We see that there are no values that σ_1 can take to satisfy both this specification and the one attached to Sib(John,George). Therefore the two conditions are not reconcilable.

Formally, we have

Definition D2.2 (***Assignment***) Let P be a program. Let $\Sigma = \{\sigma_1,...,\sigma_m\}$ be the set of S-constants in P, D be the set of constants in P, and the range of σ_i in P be r_{σ_i}.

An ***assignment*** is a set of the form $\{S_1 \subseteq V_1,...,S_n \subseteq V_n\}$ where $S_i \subseteq \Sigma$ and $V_i \subseteq D$ for $1 \le i \le n$, and $S_i \cap S_j = \emptyset$ for $1 \le i,j \le n$, $i \ne j$. We also define $\perp$ as a special assignment.

An assignment w is ***strict*** if it does not contain elements of the form $\{\sigma\} \subseteq r_\sigma$, where σ is any S-constant, and if it satisfies the condition that, for $1 \le i \le m$, $V_i \subseteq \bigcap_{\sigma \in S_i} r_\sigma$.

The ***range*** of an S-constant σ in an assignment w, written as Ran(σ,w), and the ***class*** of σ in w, written as Cla(σ,w), are defined as follows:

For the special assignment $\perp$, Cla(σ,$\perp$)=$\emptyset$, and Ran(σ,$\perp$)=$\emptyset$. For all other assignment w, if there is an element $S \subseteq V$ in w such that σ is a member of S, then Ran(σ,w)=V, and Cla(σ,w)=S, otherwise Ran(σ,w)=r_σ, Cla(σ,w)=$\{\sigma\}$. ■

Two things to note about this definition. The first is that the special assignment $\perp$ is used to represent the case when we have an unsatisfiable condition. The second is that a domain specification is an assignment but not a strict assignment. Intuitively, an element of the form $\{\sigma\} \subseteq r_\sigma$ merely expresses the initial constraint on the S-constant σ specified in the program, i.e. that σ has range r_σ and that whether σ is identical to any other S-constant is unknown. A strict assignment is one that does not contain such redundant information.

Example 2.3

Consider the program in example 2.1:

(Father(σ_1,John);{{σ_1} ⊆ {Mike,Smith,Joe}})
(Father(Paul,σ_1);{{σ_1} ⊆ {Mike,Smith,Joe}})
(Father(σ_2,George);{{σ_2} ⊆ {Dave,Mike,Smith}})
(Father(Dave,Paul);∅)
(Sib(x,y) ← Father(z,x),Father(z,y);∅)

where σ_1 and σ_2 are S-constants.

u_1 = {{σ_1} ⊆ {Mike}} and u_2 = {{σ_1,σ_2} ⊆ {Smith}} are both assignments. Ran(σ_1,u_1)={Mike} and Ran(σ_2,u_1)=r_{σ_2}={Dave,Mike,Smith} as σ_2 does not appear in u_1. Cla(σ_1,u_1)={σ_1} and Cla(σ_2,u_1)={σ_2}. On the other hand, Ran(σ_1,u_2)=Ran(σ_2,u_2) = {Smith} and Cla(σ_1,u_2)=Cla(σ_2,u_2) = {σ_1,σ_2}. ■

In defining a program with null values, we introduced extended program clauses. They are nothing more than ordered pairs of Horn clauses and domain specifications. Since domain specifications are a special kind of assignments, we can further define the following extensions:

Definition D2.3 (***Extended Clauses and Extended Goals***) Let P be a program and let w be an assignment. An ***extended clause*** is an ordered pair (C;w) where C is a clause. An ***extended goal*** is an extended clause (C;w) where C is a clause with no positive literals. ■

We now define how to combine two assignments, which is used to record the conjunction of two conditions.

Definition D2.4 (***Combining Assignments***) Let P be a program. Let $\Sigma = \{\sigma_1,...,\sigma_m\}$ be the set of S-constants in P, and let the initial range of σ_i in P be r_{σ_i} for $1 \le i \le m$. Let w_1 and w_2 be assignments. The ***combination*** of w_1 and w_2, written as w_1w_2, is defined inductively as follows:

1. If w_1 or w_2 is $\perp$, then $w_1w_2 = \perp$.
2. If neither w_1 nor w_2 are $\perp$ and w_1 (resp. w_2) is $\emptyset$, then w_1w_2 is w_2 (resp. w_1).
3. Let $w_1 = \{S_1 \subseteq V_1\}$ and $w_2 = \{S_2 \subseteq V_2\}$. If $S_1 \cap S_2 = \emptyset$ then $w_1w_2 = w_1 \cup w_2$. If $S_1 \cap S_2 \neq \emptyset$ and $V_1 \cap V_2 \neq \emptyset$ then $w_1w_2 = \{S_1 \cup S_2 \subseteq V_1 \cap V_2\}$. Otherwise ($S_1 \cap S_2 \neq \emptyset$ and $V_1 \cap V_2 = \emptyset$), $w_1w_2 = \perp$.
4. If $w_1 = \{S_{11} \subseteq V_{11},...,S_{1p} \subseteq V_{1p}\}$ where $p>1$, then $w_1w_2 = u(vw_2)$ where u is $\{S_{11} \subseteq V_{11},...,S_{1p-1} \subseteq V_{1p-1}\}$ and v is $\{S_{1p} \subseteq V_{1p}\}$.
5. If $w_1 = \{S_{11} \subseteq V_{11}\}$, and $w_2 = \{S_{21} \subseteq V_{21},...,S_{2q} \subseteq V_{2q}\}$ where $q>1$, then $w_1w_2 = (w_1u)v$ where u is $\{S_{21} \subseteq V_{21}\}$ and v is $\{S_{22} \subseteq V_{22},...,S_{2q} \subseteq V_{2q}\}$.

If $w_1w_2 \neq \perp$, then the assignments w_1 and w_2 are *compatible*, otherwise, they are *incompatible*. ■

Example 2.4
Let Σ be $\{\sigma_1,\sigma_2,\sigma_3\}$, and $r_{\sigma_1} = \{e,f,g\}$, $r_{\sigma_2} = \{f,g,h\}$, $r_{\sigma_3} = \{g,h,i\}$. Let $w_1 = \{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}, \{\sigma_3\} \subseteq \{g\}\}$, and $w_2 = \{\{\sigma_2,\sigma_3\} \subseteq \{g,h\}\}$. We want to calculate w_1w_2. According to 4 above, $w_1w_2 = u(vw_2)$ where u is $\{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\}$ and v is $\{\{\sigma_3\} \subseteq \{g\}\}$. To calculate vw_2, we use 3 above. Since $\{\sigma_3\} \cap \{\sigma_2,\sigma_3\} \neq \emptyset$, and $\{g\} \cap \{g,h\} = \{g\} \neq \emptyset$, we have $v' = vw_2 = \{\{\sigma_2,\sigma_3\} \subseteq \{g\}\}$. Similarly, using 3 again, we have $w_1w_2 = uv' = \{\{\sigma_1,\sigma_2,\sigma_3\} \subseteq \{g\}\}$. ■

We note from the examples at the beginning of this subsection that, in general, any condition that we need to record can be seen as the result of applying a sequence of substitutions to the initial constraints specified in the program. For example, the substitution $\{\sigma_2/\sigma_1\}$ results in the condition $\sigma_1 = \sigma_2$,and $\{Mike/\sigma_1\}$, results in the condition σ_1=Mike.This suggests that applying a substitution to a formula, has the same effect as attaching an assignment to that formula. Therefore, we have the following definitions:

Definition D2.5 (***Substitutions as Assignments***) Let P be a program, and $\Sigma = \{\sigma_1,...,\sigma_m\}$ be the set of S-constants in P, and the range of σ_i in P be r_{σ_i}. Let $[\,]_A$ be a mapping from substitutions to assignments. Then, let θ be a substitution, the assignment $[\theta]_A$ is called the ***equivalent assignment*** of θ, and is defined inductively as:
If $\theta = \varepsilon = \{\}$ is the identity substitution: $[\theta]_A = \emptyset$
If $\theta = \{t/v\}$, where v is a variable: $[\theta]_A = \emptyset$
If $\theta = \{t/v\}$, where v is an S-constant and t is a constant:
  If $v \in \Sigma$ and $t \in r_v$ then $[\theta]_A = \{v \subseteq \{t\}\}$. Otherwise, $[\theta]_A = \perp$.
If $\theta = \{t/v\}$, where v and t are both S-constants:
  If $v \in \Sigma$, $t \in \Sigma$, and $r_v \cap r_t \neq \emptyset$ then $[\theta]_A = \{\{v,t\} \subseteq r_v \cap r_t\}$. Otherwise, $[\theta]_A = \perp$.
If $\theta = \{t_1/v_1,...,t_n/v_n\}$ where $n>1$: $[\theta]_A = [\lambda]_A[\mu]_A$ where λ is $\{t_1/v_1\}$ and μ is $\{t_2/v_2,...,t_n/v_n\}$
If $[\theta]_A \neq \perp$ then we say that θ is ***applicable*** (w.r.t. P). Otherwise, it is inapplicable. Also, we say that $[\theta]_A$ is an assignment *expressible* by the substitution θ. ■

We see that if a substitution contains no substitutions for S-constant, then its equivalent assignment is $\emptyset$, which essentially represents the condition "true", i.e. "no additional restriction on the values of the S-constants". The reason we impose an ordering on the S-constants is that the two forms of substitutions $\{\sigma_i/\sigma_j\}$ and $\{\sigma_j/\sigma_i\}$ both express the condition that σ_1 and σ_2 are equal, and the ordering allows us to consistently choose one of them. Also, we note that given a substitution θ if $[\theta]_A \neq \perp$ then $[\theta]_A$ is strict.

Example 2.5
Let Σ be $\{\sigma_1,\sigma_2,\sigma_3\}$, and $r_{\sigma_1} = \{e,f,g\}$, $r_{\sigma_2} = \{f,g,h\}$, $r_{\sigma_3} = \{g,h,i\}$. Consider the substitution $\theta = \{\sigma_1/x, b/y, \sigma_2/\sigma_1, g/\sigma_3\}$. $[\theta]_A = [\mu_1]_A[\mu_2]_A[\mu_3]_A[\mu_4]_A$ where $\mu_1 = \{\sigma_1/x\}$, $\mu_2 = \{b/y\}$, $\mu_3 = \{\sigma_2/\sigma_1\}$, and $\mu_4 = \{g/\sigma_3\}$. Since μ_1 and μ_2 are all variable substitutions, $[\mu_1]_A = [\mu_2]_A = \emptyset$. Also, $r_{\sigma_1} \cap r_{\sigma_2} = \{e,f,g\} \cap \{g,h,i\} = \{f,g\} \neq \emptyset$, hence $[\mu_3]_A = \{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\}$. And since $g \in r_{\sigma_3}$, $[\mu_4]_A = \{\{\sigma_3\} \subseteq \{g\}\}$. So, $[\theta]_A = \emptyset\emptyset\{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\}\{\{\sigma_3\} \subseteq \{g\}\} = \{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}, \{\sigma_3\} \subseteq \{g\}\}$ ■

A dual of the above mapping is the following:

Definition D2.6 (***Assignment as Substitution***) Let $[\,]_S$ be a mapping from strict assignments to substitutions. Then, let $w=\{S_1 \subseteq V_1,...,S_n \subseteq V_n\}$ be a strict assignment, the substitution $[w]_S$ is called the ***equivalent substitution*** for w, and is defined inductively as follows:
$[\emptyset]_S=\varepsilon$, the identity substitution.
V_1 is a singleton {c} where c is some constant. Then $[w]_S=\{c/\tau \mid \tau \in S_1\} \cup [w-\{S_1 \subseteq V_1\}]_S$
V_1 is not a singleton, and S_1 is the set $\{\sigma_{i_1},...,\sigma_{i_m}\}$, where $i_j<i_k$ for j<k. Then
$[w]_S=\{\sigma_{i_j}/\sigma_{i_1} \mid 1\le j<m\} \cup [w-\{S_1 \subseteq V_1\}]_S$ ■
In the above definition, since $S_i \cap S_j=\emptyset$ for $i\neq j$, the order in which we convert the elements in an assignment w is irrelevant.

Example 2.6
Let us find the equivalent substitution for the assignment in the previous example.
$w = \{\{\sigma_1,\sigma_2\} \subseteq \{f,g\},\{\sigma_3\} \subseteq \{g\}\}$
$[w]_S = \{\sigma_2/\sigma_1\} \cup [\{\{\sigma_3\} \subseteq \{g\}\}]_S = \{\sigma_2/\sigma_1\} \cup \{g/\sigma_3\} = \{\sigma_2/\sigma_1, g/\sigma_3\}$.
■

Among all the possible assignments that we can form using the S-constants and constants, not every one is expressible by a substitution. Notably, any assignment w that contains an element of the form $S \subseteq V$, where S is a singleton, say $\{\sigma\}$, and V is the set r_σ (i.e. w is not a strict assignment) can not be an equivalent assignment of a substitution. To see this, we note that only two kinds of substitutions would permit an S-constant, σ, to appear in an equivalent assignment, namely, substitution of σ by a constant, c, or by another S-constant, τ. In the first case, V would become {c}, and in the second case, S would be $\{\sigma,\tau\}$.

In defining a model semantics of a logic program with null values, we only need to consider assignments that are expressible by substitutions:

Definition D2.7 (***Assignment Universe***) Let P be a program. The ***assignment universe*** of P is the set $\{[\theta]_A \mid \theta$ is applicable w.r.t. P$\}$. ■

From now on, unless specified otherwise, an assignment always refers to one that is in the assignment universe.

Finally, we have the definition of the Herbrand Base with null values:

Definition D2.8 (***Herbrand Base with Null Values***) Let P be a program, qHB(P) be its quasi-Herbrand Base, and W be its assignment universe.
The ***Herbrand Base with null values*** for program P, HBNV(P), is the set of ordered-pairs (A;w) where $A \in$ qHB(P), $w \in W$.
We call elements in HBNV(P) *(null-)extended atoms.* ■

Note that for the case in which the program does not have any null values, i.e. when $\Sigma=\emptyset$, no substitution contains S-constants, and therefore, $[\theta]_A=\emptyset$ for all applicable substitutions θ. $W=\{\emptyset\}$. Also, qHB reduces to the classical Herbrand Base, and every element in HBNV is of the form $(A;\emptyset)$. Since there are no S-constants, $(A;\emptyset)$ represents the atom A without any condition attached, that is, HBNV is isomorphic to the classical Herbrand Base.

Below is an example that illustrates some of the definitions in this section.

Example 2.7
Consider the program P which contains the following :

$(F(\sigma_1,e);\{\{\sigma_1\} \subseteq \{e,f,g\}\})$
$(F(\sigma_2,f);\{\{\sigma_2\} \subseteq \{f,g,h\}\})$
$(S(x,y) \leftarrow F(z,x),F(z,y);\emptyset)$

Then ${}_NU_P$ is the set $\{e,f,g,h,\sigma_1,\sigma_2\}$. So,

$$qHB(P)=\{F(e,e),F(e,f),F(e,g),F(e,h),F(e,\sigma_1),F(e,\sigma_2),F(f,e),...,F(f,\sigma_1),F(f,\sigma_2),...,$$
$$F(\sigma_1,e),...,F(\sigma_1,\sigma_1),F(\sigma_1,\sigma_2),F(\sigma_2,e),...,F(\sigma_2,\sigma_1),F(\sigma_2,\sigma_2),$$
$$S(e,e),...,S(f,e),...,S(\sigma_1,e),...,S(\sigma_2,e),...,S(\sigma_2,\sigma_2)\}$$

and the assignment universe is:

$$W=\{\varnothing,\{\{\sigma_1\}\subseteq\{e\}\},\{\{\sigma_1\}\subseteq\{f\}\},\{\{\sigma_1\}\subseteq\{g\}\},\{\{\sigma_2\}\subseteq\{f\}\},\{\{\sigma_2\}\subseteq\{g\}\},\{\{\sigma_2\}\subseteq\{h\}\},$$
$$\{\{\sigma_1\}\subseteq\{e\},\{\sigma_2\}\subseteq\{f\}\},\{\{\sigma_1\}\subseteq\{e\},\{\sigma_2\}\subseteq\{g\}\},\{\{\sigma_1\}\subseteq\{e\},\{\sigma_2\}\subseteq\{h\}\},$$
$$\{\{\sigma_1\}\subseteq\{f\},\{\sigma_2\}\subseteq\{f\}\},\{\{\sigma_1\}\subseteq\{f\},\{\sigma_2\}\subseteq\{g\}\},\{\{\sigma_1\}\subseteq\{f\},\{\sigma_2\}\subseteq\{h\}\},$$
$$\{\{\sigma_1\}\subseteq\{g\},\{\sigma_2\}\subseteq\{f\}\},\{\{\sigma_1\}\subseteq\{g\},\{\sigma_2\}\subseteq\{g\}\},\{\{\sigma_1\}\subseteq\{g\},\{\sigma_2\}\subseteq\{h\}\},$$
$$\{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\},\{\{\sigma_1,\sigma_2\}\subseteq\{f\}\},\{\{\sigma_1,\sigma_2\}\subseteq\{g\}\}\}$$

We will not list all the elements of HBNV(P) here. As we will see later, some of the elements of HBNV(P) that will be in the "model" of the program are: $(F(\sigma_1,e);\varnothing)$, $(F(\sigma_2,f);\varnothing)$, $(S(e,e);\varnothing)$, $(S(f,f);\varnothing)$, $(S(e,f);\{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\})$, $(S(f,e);\{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\})$. ■

Given an atom A in the Herbrand Base of a definite program (without null values), we would like to know whether A is derivable from the program. Similarly, given an extended atom (A;w) in the HBNV of a program P, we would like to know whether A is derivable from P under the condition represented by w. We will see other aspects in which the relationship between HBNV and programs with null values are analogous to that between Herbrand Base and definite programs without null values.

2.2. Unification

Unification will be a basic concept used in subsequent discussions. First, we define what we mean by a set of atoms being unifiable when they contain S-constants.

Definition D2.9 (***Unifier and MGU***) A set of atoms $T = \{A_1,...,A_n\}$ is unifiable if there is a substitution θ such that $A_i\theta = A_j\theta$ for $1\le i,j\le n$. θ is called a ***unifier*** of T. θ is a ***most general unifier (mgu)*** of T, if for each unifier ρ of T, there is a substitution μ such that $\rho=\theta\mu$. ■

Example 2.8
Consider the set $\{P(\sigma,f(y)),P(b,x)\}$. If we apply the substitution $\rho=\{f(a)/x,a/y,b/\sigma\}$ to the two atoms in the set, then they both become P(b,f(a)). Therefore, ρ is a unifier for the set. However, ρ is not a most general unifier. An example of an mgu for this set is $\theta=\{f(y)/x,b/\sigma\}$. Note that $\rho=\theta\{a/y\}$. ■

Unifiers for extended atoms can now be defined.

Definition D2.10 (***Unifiers for Extended Atoms***) A set of extended atoms $T = \{(A_1;u_1),...,(A_n;u_n)\}$ is unifiable if there is a substitution θ such that θ is a unifier of the set of atoms $t = \{A_1,...,A_n\}$ and $[\theta]_A u_1...u_n\neq\perp$. θ is called a unifier of T. A unifier θ of T is an mgu iff θ is an mgu of t. ■

Example 2.9
$\{(P(g(\sigma_1),f(y));\{\{\sigma_1\}\subseteq\{a,b,c\}\}),(P(g(\sigma_2),x);\{\{\sigma_2\}\subseteq\{b,c,d\}\})\}$ is unifiable with the mgu $\theta=\{f(y)/x,\sigma_2/\sigma_1\}$, since $[\theta]_A\{\{\sigma_1\}\subseteq\{a,b,c\}\}\{\{\sigma_2\}\subseteq\{c,d,e\}\}=\{\{\sigma_1,\sigma_2\}\subseteq\{c\}\}\neq\perp$.
On the other hand, $\{(P(\sigma_1,f(y));\{\{\sigma_1\}\subseteq\{a,b,c\}\}),(P(\sigma_2,x);\{\{\sigma_2\}\subseteq\{e,f,g,\}\})\}$ is not unifiable since $[\theta]_A\{\{\sigma_1\}\subseteq\{a,b,c\}\}\{\{\sigma_2\}\subseteq\{e,f,g\}\}=\perp$. ■

The unification algorithm for atoms without S-constants can be adapted to handle S-constants. First, we need some definitions ([10]).

Definition D2.11 (***Term Equations and Solutions***)
A ***term equation*** is an equation of the form s=t where s and t are terms.
A substitution θ is called a ***unifier*** of a term equation s=t iff $s\theta$ and $t\theta$ are identical. A substitution θ is called a unifier of a set of term equations S iff θ is a unifier of every term equation in S.
Two sets of term equations are called ***equivalent*** iff they have the same unifiers.
A (possibly empty) set of term equations is called ***solved*** iff it is of the form $\{v_1=t_1,...,v_n=t_n\}$, where v_i's are distinct variables or S-constants, and none of them occurs in a term t_j. Also, if v_i is an S-constant, then t_i is either another S-constant or a constant. ■

A solved set of term equations $\{v_1=t_1,...,v_n=t_n\}$ determines a unique substitution $\{t_1/v_1,...,t_n/v_n\}$. Clearly, this substitution is a most general unifier of the set of equations. In order for two atoms to be unifiable, they must have the same predicate symbol. Finding the mgu of two atoms, $P(s_1,...,s_n)$ and $P(t_1,...,t_n)$ is the same as finding the mgu of the set of term equations $s_1=t_1,\ldots,s_n=t_n$
The following unification algorithm is adapted from [10].

UNIFICATION ALGORITHM

Non-deterministically choose from the set of equations an equation of a form below and perform the associated action. (In the following, f and g are two different function symbols, c and d are two different constants, v is either a variable or an S-constant, and all the others are terms.)

(1) $f(s_1,...,s_n)=f(t_1,...,t_n)$:
replace by the equation $s_1=t_1,...,s_n=t_n$

(2) $c=d$ or $f(_1,...,s_n)=g(t_1,...,t_n)$:
halt with failure.

(3) $c=c$ or $v=v$:
delete the equation

(4) $t=v$ where t is neither a variable nor an S-constant :
replace by the equation $v=t$

(5) $t=v$ where t is an S-constant and v is a variable:
replace by the equation $v=t$

(6) $v=t$ where v is different from t, and v has another occurrence in the set of equations :
if v appears in t then halt with failure
if v is an S-constant and t is neither a constant nor an S-constant then halt with failure
if v is the S-constant σ_i and t is the S-constant σ_j and i>j, then replace by the equation $t=v$
otherwise perform the substitution {t/v} in every other equation

Example 2.10

To find the mgu of the two atoms $P(g(\sigma_1),f(y))$ and $P(g(\sigma_2),x)$ we solve the set of equations $\{g(\sigma_1)=g(\sigma_2),f(y)=x\}$.

1. Choosing the first equation and using (1) above, we replace it by the equation $\sigma_1=\sigma_2$.
2. By using (4) above, we replace the second equation by $x=f(y)$.
3. The set of equations is now $\{\sigma_1=\sigma_2,x=f(y)\}$, and it is solved.

The corresponding mgu is $\{\sigma_2/\sigma_1,f(y)/x\}$. ■

It is easy to augment the proof in [10] to show that the following theorem is true.

Theorem T2.1 (***Unification Theorem***) Let T be a finite set of term equations. If T is solvable, then the unification algorithm terminates and gives an mgu for T. If T is not solvable, then the unification algorithm terminates and reports this fact.

Proof:

As far as the unification algorithm is concerned, S-constants are identical to variables except for, (a) the ordering among the S-constants, and (b) the requirement that an S-constant be unified only with S-constants or constants. In the proof of the original algorithm, the order of the variables is not important. Also, since we define substitution such that an S-constant can only be substituted by either an S-constant or a constant, point (b) is exactly what is needed to make the mgu produced by the algorithm a proper substitution. Therefore, the proof holds for our modified algorithm regardless of the presence of S-constants. *Q.E.D.*

2.3. State and Model State

A program with null values can be viewed as a compact way of representing a set of possible worlds. For instance, if P has the following extended program clauses: $(P(\sigma_1);\{\{\sigma_1\}\subseteq\{a,b\}\})$, $(Q(\sigma_2);\{\{\sigma_2\}\subseteq\{c,d\}\})$, and $(R(e);\emptyset)$; which means that either a or b is in relation P, either c or d is in relation Q, and e is in relation R. We can view P as representing the four possible worlds: P_1: {P(a),Q(c),R(e)}, P_2: {P(a),Q(d),R(e)}, P_3: {P(b),Q(c),R(e)}, P_4: {P(b),Q(d),R(e)}. These are obtained from P by assigning to each null value one of the constant in the range of the null value. Each possible world can therefore be identified with a particular substitution. For instance, P_1 can be identified with the substitution "a for σ_1 and c for σ_2", and P_2 the substitution "a for σ_1 and d for σ_2", etc.

This above discussion motivates the following definitions.

Definition D2.12 (***S-Mapping***) Let P be a program, and let $\Sigma=\{\sigma_1,...,\sigma_n\}$ be the set of S-constants appearing in P and *D* be the set of constants appearing in P. A function V from Σ to *D* is an ***S-mapping*** iff V maps each element σ_i of Σ to a constant c_{ij} in the initial range of σ_i. We write an S-mapping V as a set of values $\{a_1,...,a_n\}$ to represent the fact that $V(\sigma_1)=a_1$, $V(\sigma_2)=a_2,...,V(\sigma_n)=a_n$. ■

Definition D2.13 (***S-Mapping Substitution and Possible Worlds***) Let P be a program and Σ be the set of S-constants appearing in P. Let V be an S-mapping, then the ***substitution corresponding to an S-mapping*** V is the substitution $\{V(\sigma_1)/\sigma_1,...,V(\sigma_n)/\sigma_n\}$, usually written as ρ_V.
Let S be a set of extended clauses, the ***image*** of S in the S-mapping V, written as S_V, is the set of first order formula: $\{R\rho_V \mid (R;w)\in S \wedge w[\rho_V]_A\neq\perp\}$ The ***possible world*** represented by P and the S-mapping V, is the image of P in V, P_V. ■

In a program without null values, any subset of the Herbrand base is a Herbrand interpretation of the program. Subsets of the HBNV can play a similar role if we extend the notion of possible world to these subsets. Consider a subset, I, of HBNV. An element, (A;w), in I represents the atom A with the condition w. To compute the image of this element in a possible world represented by the S-mapping V, we first see whether w is satisfied in this world (i.e. whether $w[\rho_V]_A$ is $\perp$). If it is satisfied ($w[\rho_V]_A\neq\perp$) then the image is $A\rho_V$, otherwise, the element has no counterpart in this particular world. The image of I itself is simply the collection of the images of all the elements in I, which is the set $\{A\rho_V \mid (A;w)\in I \wedge w[\rho_V]_A\neq\perp\}$. We see that this image is a set of ground atoms, and can therefore serve as a Herbrand interpretation for the program P_V.

For example, consider the program P:

$(P(\sigma_1);\{\{\sigma_1\}\subseteq\{a,b\}\})$
$(Q(\sigma_2);\{\{\sigma_2\}\subseteq\{c,d\}\})$
$(R(e);\varnothing)$
$(S(x,y)\leftarrow P(x),Q(y);\varnothing)$

Now consider the following subset I of HBNV(P):

$\{(P(a);\varnothing),(Q(c);\{\{\sigma_2\}\subseteq\{c\}\}),(Q(d);\{\{\sigma_2\}\subseteq\{d\}\})\}$

As before, there are four possible S-mappings, namely, $V_1=\{a,c\}$, $V_2=\{a,d\}$, $V_3=\{b,c\}$, and $V_4=\{b,d\}$. The image of P in V_1, P_{V_1}, contains the four definite clauses $\{P(a),Q(c),R(e),S(x,y)\leftarrow P(x),Q(y)\}$. And, the image of I in V_1 is $\{P(a),Q(c)\}$. If we treat this as a Herbrand interpretation, then the only atoms that evaluate to true in this interpretation are P(a) and Q(c). Therefore, the four clauses in P_{V_1} have truth values w.r.t. I, "true", "true", "false", and "false" respectively. On the other hand, in V_4, the image of I is $\{P(c),Q(d)\}$. Hence, the four clauses in P_{V_4}, i.e. $\{P(b),Q(d),R(e),S(x,y)\leftarrow P(x),Q(y)\}$, have truth values "false","true", "false", and "false". In fact, it is easy to see that, with respect to I, the second clause in P has truth value "true" in all four possible worlds, and the third and forth clauses are always false, while the first clause is true in V_1 and V_2 and false in V_3 and V_4.

The following definition is a natural outcome of this discussion.

Definition D2.14 (***States***) Let P be a program, V be an S-mapping, and S be a subset of HBNV(P). Let (C;w) be an extended clause such that $w[\rho_V]_A\neq\perp$, then the value of (C;w)in V w.r.t. S is the Herbrand interpretation of $C\rho_V$ w.r.t. the set S_V. *S is a **state** of P iff for every extended clause (C;w) in P, (C;w) has* the same value w.r.t. S in every S-mapping. ■
It is easy to see that if w is a domain specification, then $w[\rho_V]_A\neq\perp$ for any S-mapping V. Therefore, every extended program clause in a program has a value in every S-mapping w.r.t. every subset of HBNV(P). A state, then, is just a collection of "consistent" interpretations. By this we mean that a clause is interpreted consistently in every possible world, i.e. we do not want to have a clause being interpreted as "true" under one S-mapping while interpreted as "false" under another S-mapping.

For instance, the subset I in the previous discussion is not a state, but the set $\{(Q(c);\{\{\sigma_2\}\subseteq\{c\}\}),(Q(d);\{\{\sigma_2\}\subseteq\{d\}\})\}$ is a state, as well as the set $\{(P(\sigma_1);\varnothing),(Q(c);\{\{\sigma_2\}\subseteq\{c\}\}),(Q(d);\{\{\sigma_2\}\subseteq\{d\}\})\}$.

The definition of state has a shortcoming which is inherent in the format of extended Herbrand base: two apparently different subsets of HBNV can represent the same state, i.e. they have the same image in all

S-mappings. For instance, let us continue our previous example, where the S-mappings are $V_1 = \{a,c\}$, $V_2 = \{a,d\}$, $V_3 = \{b,c\}$, and $V_4 = \{b,d\}$, and consider the two sets $I_1 = \{(P(a);\{\{\sigma_1\} \subseteq \{a\}\}),(P(b);\{\{\sigma_1\} \subseteq \{b\}\})\}$ and $I_2 = \{(P(\sigma_1);\emptyset)\}$. Although they look quite different, their "image" in every S-mapping is the same, i.e. both are $\{P(a)\}$ in V_1 and V_2, and both are $\{P(b)\}$ in V_3 and V_4. Therefore, there is no practical difference between the two.

To deal with this shortcoming, we define the following relation

Definition D2.15 (***Equivalence Relation for States***) Let P be a program and let S and T be two subsets of HBNV(P). We say that S and T are *equivalent*, written as $S \equiv_m T$, iff for every S-mapping V, $S_V = T_V$. ■

It is obvious that $\equiv_m$ is an equivalence relation. Similar to the definition of Herbrand models, we have the following.

Definition D2.16 (***Model State***) Let P be a program and I be a state of P. I is a ***model state*** of an extended clause (C; w) iff for every S-mapping V, if $w[\rho_V]_A \neq \perp$ then I_V is a Herbrand model of $C\rho_V$. I is a ***model state*** of P iff I is a model state of every extended clause in P, and for every state J such that $J \equiv_m I$, $J \subseteq I$. ■

The last condition in the above definition can be understood in another way. Since $\equiv_m$ is an equivalence relation, the set of all the states can be divided into equivalence classes. Given any state I, suppose that I belongs to the equivalence class E, then it is easy to see that the union of all the elements in E is itself a state, and moreover, this union is also equivalent to I and hence also a member of E. That is, $(\bigcup_{J \in E} J) \equiv_m I$, and hence $(\bigcup_{J \in E} J) \in E$. Let us call this union M. From the definition of model state of extended clauses, if I is the model state of an extended clause, then every member in E is also a model state of that clause. Therefore, if I is a model state of all the extended clauses in a program P, then so is M, and since all other states in E are subsets of M, M is a model state of P.

The following proposition concerns the existence of such a model state.

Proposition P2.1 Let P be a program, and let $V_1,...,V_n$ be all the possible S-mappings. If $M_1,...,M_n$ are Herbrand models of P_{V_1} ,..., P_{V_n}, respectively, then there exists a unique model state S of P such that $S_{V_1}=M_1$,..., $S_{V_n}=M_n$.

Proof:

We prove this by constructing S. Let J_i be the set $\{(A;[\rho_{V_i}]_A) \mid A \in M_i\}$, then the image of J_i in V_i is exactly the set M_i. Let J be the set $\bigcup_{1 \le i \le n} J_i$, then J is a subset of HBNV(P) and J is a model state for every extended clause in P. Therefore, the set $S = \bigcup_{K \equiv_m J} K$ is a model state of P. Also, S is unique, since for any other state S', if $S'_{V_i}=M_i$ then $S' \equiv_m J$ and hence $S' \subseteq S$. *Q.E.D.*

Next, we introduce the concept of a minimal model state. Given a program P, for any S-mapping V, P_V is a definite program. Hence for each V, P_V has a unique least model. By proposition P2.1, there is a unique model state whose image in V is this least model.

Furthermore, as we see below, just as in the case of definite programs, this model state is the intersection of all model states of P.

Proposition P2.2 (***Least Model State***) Let P be a program with null values. There is a unique model state S of P such that S_V is the least Herbrand model of P_V for any S-mapping V. Furthermore, $S=\bigcap\{K \mid K \text{ is a model state of P}\}$.

Proof:

Let V_1 ,..., V_n be all the S-mappings and let M_i be the least model in V_i. From proposition P2.2 we know that there is a unique model state S such that $S_{V_i}=M_i$ for $1 \le i \le n$. Let $S'=\bigcap\{K \mid K \text{ is a model state of P}\}$. We need to prove that $S=S'$.

Since S is a model state of P, $S'_{V_i}=\bigcap\{K_{V_i} \mid K \text{ is a model state}\} \subseteq S_{V_i}=M_i$. On the other hand, since M_i is the least Herbrand model of P_{V_i} and K_{V_i} are all Herbrand models of P_{V_i}, $M_i \subseteq S'_{V_i}$. Therefore, $S'_{V_i}=M_{V_i}$ and hence $S' \equiv_m S$, so $S' \subseteq S$.

For any model state K, K can be constructed from the Herbrand models K_{V_1} ,..., K_{V_n} as seen in the proof of proposition P2.2. $S_{V_i}=M_i \subseteq K_{V_i}$ for $1 \le i \le n$. Let D_i be $K_{V_i} - M_i$. Then for any J such that $J \equiv_m S$,

the set $J \cup (\bigcup_{1 \le i \le n} \{(A;[\rho_i]_A) \mid A \in D_i\})$ is equivalent to K. Therefore, $J \subseteq K$, and since S is the union of all such J, $S \subseteq K$. Hence $S \subseteq \bigcap \{K \mid K \text{ is a model state}\}$, i.e. $S \subseteq S'$. *Q.E.D.*

Example 2.11
Consider the following program P:

$(F(\sigma_1,e);\{\{\sigma_1\} \subseteq \{e,f,g\}\})$
$(F(\sigma_2,f);\{\{\sigma_2\} \subseteq \{f,g,h\}\})$
$(S(x,y) \leftarrow F(z,x),F(z,y);\emptyset)$

It is easy to see that the state

$M=\{ (F(\sigma_1,e);\emptyset),(F(\sigma_2,f);\emptyset),(S(e,e);\emptyset),(S(f,f);\emptyset),$
$(S(e,f);\{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\}),(S(f,e);\{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\})\}$

is a model state of all the extended clauses in P. For instance, consider the S-mapping $V_1=\{e,f\}$. $P_{V_1} = \{F(e,e), F(f,f), S(x,y) \leftarrow F(z,x),F(z,y)\}$ and $M_{V_1} = \{F(e,e),F(f,f),S(e,e),S(f,f)\}$ which is a Herbrand model of P_{V_1}. Furthermore, we see that if we remove any element from M_{V_1} then it is no longer a model of P_{V_1}, i.e. M_{V_1} is the least model of P_{V_1}. This result applies to all other eight S-mappings. Therefore, we can conclude that the largest equivalent state of M is the least model state of P. The part of this model state that is equivalent to the extended atom $(F(\sigma_1,e);\emptyset)$ is shown in Fig. 1. ■

$\{ (F(\sigma_1,e);\emptyset),(F(e,e);\{\{\sigma_1\} \subseteq \{e\}\}),(F(f,e);\{\{\sigma_1\} \subseteq \{f\}\}),(F(g,e);\{\{\sigma_1\} \subseteq \{g\}\}),$
$(F(\sigma_1,e);\{\{\sigma_2\} \subseteq \{f\}\}),(F(\sigma_1,e);\{\{\sigma_2\} \subseteq \{g\}\}),(F(\sigma_1,e);\{\{\sigma_2\} \subseteq \{h\}\}),$
$(F(e,e);\{\{\sigma_1\} \subseteq \{e\},\{\sigma_2\} \subseteq \{f\}\}),(F(f,e);\{\{\sigma_1\} \subseteq \{f\},\{\sigma_2\} \subseteq \{f\}\}),$
$(F(g,e);\{\{\sigma_1\} \subseteq \{g\},\{\sigma_2\} \subseteq \{f\}\}),(F(e,e);\{\{\sigma_1\} \subseteq \{e\},\{\sigma_2\} \subseteq \{g\}\}),$
$(F(f,e);\{\{\sigma_1\} \subseteq \{f\},\{\sigma_2\} \subseteq \{g\}\}),(F(g,e);\{\{\sigma_1\} \subseteq \{g\},\{\sigma_2\} \subseteq \{g\}\}),$
$(F(e,e);\{\{\sigma_1\} \subseteq \{e\},\{\sigma_2\} \subseteq \{h\}\}),(F(f,e);\{\{\sigma_1\} \subseteq \{f\},\{\sigma_2\} \subseteq \{h\}\}),$
$(F(g,e);\{\{\sigma_1\} \subseteq \{g\},\{\sigma_2\} \subseteq \{h\}\}),(F(\sigma_2,e);\{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\}),$
$(F(\sigma_2,e);\{\{\sigma_1,\sigma_2\} \subseteq \{f\}\}),(F(\sigma_2,e);\{\{\sigma_1,\sigma_2\} \subseteq \{g\}\}),$
$(F(f,e);\{\{\sigma_1,\sigma_2\} \subseteq \{f\}\}),(F(g,e);\{\{\sigma_1,\sigma_2\} \subseteq \{g\}\})\}$

Fig. 1 (Example 2.11) Part of the Least Model State of Program P.

The following definition is analogous to the definition for "logical consequence" in first order logic.

Definition D2.17 ($\models_N$) Let P be a program. If every model state M of P is a model state of an extended clause (C; w), then we say that $P \models_N (C; w)$. ■

We denote the unique model state in the above proposition as M_P^N. The following theorem argues strongly for using M_P^N as the declarative semantics for P.

Theorem T2.2 Let P be a program and (A;u) be an element in HBNV(P). Then $P \models_N (A;u)$ iff $(A;u) \in M_P^N$.

Proof:

$P \models_N (A;u)$
iff for any model state S of P, and for all S-mapping V such that $u[\rho_V]_A \neq \perp$, $S_V \models A\rho_V$.
iff for any model state S of P, and for all S-mapping V such that $u[\rho_V]_A \neq \perp$, $A\rho_V \in S_V$.
($\because A\rho_V$ is ground)
iff for any model state S of P, $\bigcup_V \{(A\rho_V;[\rho_V]_A)\} \subseteq S$.
iff for any model state S of P, $(A;u) \in S$. ($\because \{(A;u)\} =_m \bigcup_V \{(A\rho_V;[\rho_V]_A)\}$)
iff $(A;u) \in M_P^N$. (proposition P2.2) *Q.E.D.*

3. Fixpoint Semantics

In a definite program without null values, we can calculate, in a bottom-up fashion, the set of all atoms that are derivable from the program. Let us call this set I, and we start by setting I to the empty set. We obtain new atoms by applying the clauses in the program in the following way. Given a clause in the program $H \leftarrow$ body, if we can find a substitution θ such that when we apply θ to body, all the atoms in the resulting bodyθ are in set I, then we say that $H\theta$ is one-step derivable from the set I. After we have found all such atoms, we add them to I. We repeat this process of a one-step derivation until we cannot find any new atoms to add to I.

The above description of one-step derivation is formalized as an operator defined below, and the process of finding the set of atoms derivable from the program is the same as finding the fixpoint of this operator.

Definition D3.1 (***Fixpoint Operator for definite Programs [12]***) Let P be a definite program without any S-constant, and I be a subset of its Herbrand base. The fixpoint operator of P, represented by T_P, is defined by $T_P(I) = \{A \mid A \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in P,and $\{B_1,\ldots,B_n\} \subseteq I\}$ ■

We can follow the same approach for programs with null values, except that we need to take care of the incomplete information. This means that not only must we find new atoms by applying program clauses, we also have to find the proper assignments to attach to these atoms. First, we need to define ground instances of an extended clause.

Definition D3.2 (***Ground Instances of Extended Clauses***) Let P be a program, (C;d) be an extended program clause in P, and $\theta = \theta_v\theta_s$ be a substitution, where θ_v is a variable substitution, and θ_s is an applicable substitution w.r.t. P. Then $(C\theta;[\theta]_A)$ is a ground instance of (C;d) iff $C\theta$ contains no variable. Gnd(P) is the set of all the ground instances of the extended clauses in P. ■

Note that d in the above definition is a domain specification, and as such, is of the form $\{\{\sigma_1\} \subseteq D_1,\ldots,\{\sigma_n\} \subseteq D_n\}$, where for $1 \le i \le n$, D_i is the initial range of σ_i. Since d is not an element of the assignment universe, we can not define the ground instance of (C;d) to be $(C\theta;d[\theta]_A)$. However, since d specifies the same condition as the empty assignment $\emptyset$, $(C\theta;[\theta]_A)$ is all we need.

The following property is useful.

Proposition P3.1 Let P be a program, and C=(C;d) be an extended program clause in P. If $(C\theta;[\theta]_A)=(A\leftarrow B_1,\ldots,B_n;w)$ is a ground instance of C and if for some S-mapping V, $w[\rho_V]_A \neq \perp$, then $(A\rho_V \leftarrow B_1\rho_V,\ldots,B_n\rho_V;[\rho_V]_A)$ is also a ground instance of C.

Proof:

Since $w[\rho_V]_A=[\theta]_A[\rho_V]_A \neq \perp$, then $[\theta\rho_V]_A=[\rho_V]_A$. (A general property of ρ_V) *Q.E.D.*

In defining our fixpoint operator, we need the following closure operator.

Definition D3.3 (***Equivalence Closure***) Let P be a program, and I be a subset of HBNV(P). The ***equivalence closure*** (or *E-closure*) of I, equ(I), is the set $\bigcup_{J \sim I} J$. ■

Since the only subset of the HBNV that is equivalent to the empty set $\emptyset$ is the empty set itself, we have $equ(\emptyset)=\emptyset$. In general, equ(I) is much larger than I, as illustrated by the following example.

Example 3.1

Consider the program in example 2.11. Let $I=\{(F(\sigma_1,e);\emptyset)\}$. Then Figure 1 is the E-closure of I. ■

Definition D3.4 (***Fixpoint Operator***) Let P be a program ${}_NT_P$ is a mapping from the power set of HBNV(P) to the power set of HBNV(P). If $I \subseteq$ HBNV(P), then ${}_NT_P(I)=equ(I) \cup T_P(I)$ where

$T_P(I)=\{(A;w) \mid (A \leftarrow B_1,\ldots,B_n;w) \in Gnd(P)$,

and there are $(B_1';w_1),\ldots,(B_n';w_n)$ in I, such that $w=w_1\ldots w_n \neq \perp$ and $B_i'[w]_S=B_i$ for $1 \le i \le n\}$

We also define ${}_NT_P\uparrow n$ and $T_P\uparrow n$ inductively as

${}_NT_P\uparrow 0=\emptyset$ ${}_NT_P\uparrow n={}_NT_P({}_NT_P\uparrow(n-1))$ for $n>0$.
$T_P\uparrow 0=\emptyset$ $T_P\uparrow n=T_P(T_P\uparrow(n-1))$ for $n>0$.

■

Both T_P and ${}_NT_P$ are continuous (see Appendix), and since the power set of HBNV(P) is a complete lattice, this means that ${}_NT_P\uparrow\omega$ (resp. $T_P\uparrow\omega$) is the least fixpoint of ${}_NT_P$ (resp. T_P).

As noted before, equ(I) is expensive to calculate. However, as will be clear in the rest of the section, the inclusion of equ in the definition of the ${}_NT_P$ operator is mainly to facilitate the proof of the next proposition. In fact, we shall see later that to calculate the least fixpoint of ${}_NT_P$, we can first calculate the least fixpoint of T_P and then take its E-closure.

Example 3.2
Recall example 2.11, where the following program P is given:

$(F(\sigma_1,e);\{\{\sigma_1\}\subseteq\{e,f,g\}\})$
$(F(\sigma_2,f);\{\{\sigma_2\}\subseteq\{f,g,h\}\})$
$(S(x,y)\leftarrow F(z,x),F(z,y);\varnothing)$

We will only consider T_P here.

$T_P\uparrow 1=\{(F(\sigma_1,e);\{\{\sigma_1\}\subseteq\{e,f,g\}\}),(F(\sigma_2,f);\{\{\sigma_2\}\subseteq\{f,g,h\}\})\}$.

$T_P\uparrow 2=T(T_P\uparrow 1)=T_P\uparrow 1\cup\{(S(e,e);\varnothing),(S(f,f);\varnothing),(S(e,f);\{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\})\}$

For any $k>2$, $T_P\uparrow k=T_P\uparrow 2$. Therefore, $T_P\uparrow 2$ is the fixpoint. ■

The next proposition states that the model state of a program is identical to the pre-fixpoints of ${}_NT_P$.

Proposition P3.2 Let P be a program. I is a model state of P iff ${}_NT_P(I)\subseteq I$.
Proof:
Only if: I is a model state of P.
We first prove that $T_P(I)\subseteq I$. Then by the definition of model state, $equ(I)\subseteq I$, and hence ${}_NT_P(I)\subseteq I$.
If an element (A;w) is in $T_P(I)$, then there is is a ground instance of a clause in P, $(A\leftarrow B_1,...,B_n;w_0)$, and there are n elements, $(B_i;w_n)$ $1\le i\le n$, in I such that $w=w_0w_1...w_n\neq\perp$. This implies that there is an S-mapping such that $w_i[\rho_V]_A\neq\perp$ for $0\le i\le n$. Since I is a model state of every extended clause in P, and since $w_0[\rho_V]_A\neq\perp$, I_V is a Herbrand model of $A\rho_V\leftarrow(B_1,...,B_n)\rho_V$. Since $w_i[\rho_V]_A\neq\perp$ for $1\le i\le n$, $B_i\rho_V$ is in I_V. Therefore, $A\rho_V$ must be in I_V. By the definition of model state, we know that I contains all elements of HBNV(P) whose image is $A\rho_V$, i.e.

$$\forall(A';w')\in HBNV(P)\,(w'[\rho_V]_A\neq\perp\wedge A'\rho_V=A\rightarrow(A';w')\in I) \qquad (*)$$

In particular, if we let (A';w') be (A;w), then all the premises of (*) would be satisfied, and hence $(A;w)\in I$. That is, $T_P(I)\subseteq I$.
if: ${}_NT_P(I)\subseteq I$
We prove that if $T_P(I)\subseteq I$ then I is a model state of all the extended clauses of P. And since $equ(I)\subseteq I$, $J\subseteq I$ for any $J\equiv_m I$. Then, we would have I as a model state.
Let $(A\leftarrow B_1,...,B_n;w_0)$ be a ground instance of an extended clause C in P. Let V be an S-mapping such that $w_0[\rho_V]_A\neq\perp$. That is, $A\rho_V\leftarrow(B_1,...,B_n)\rho_V$ is a ground instance of C_V. Suppose there are n elements, $(B_i;w_i)$ $1\le i\le n$, in I, such that $w_i[\rho_V]_A\neq\perp$, i.e. $B_i\rho_V\in I_V$.
$w_i[\rho_V]_A\neq\perp$, for $0\le i\le n$, implies that if $w=w_0w_1...w_n$, then $w[\rho_V]_A\neq\perp$. By the definition of T_P, $(A;w)\in T_P(I)$, and hence, $(A;w)\subseteq I$. That is, $A\rho_V$ is in I_V, therefore, I_V is a Herbrand model of this ground instance of clause C_V. This argument holds for any ground instance of C_V where C can be any extended clause in P. Therefore, I is a model state of all extended clauses in P. *Q.E.D.*

In proving the following theorem we use a property of complete lattices and fixpoint operators ([12]), namely,

Proposition P3.3 If L is a complete lattice, and T is an operator on this lattice, then the least fixpoint of T is equal to the greatest lower bound (glb) of the set $\{I\in L\mid T(I)\subseteq I\}$ ■

We will also use the fact that for any continuous (hence monotonic) operator T, $T\uparrow n\subseteq T\uparrow(n+1)$ for $n\ge 0$. This is easily proven by induction on n.

Theorem T3.1 (***Fixpoint Characterization of Least Model State***) Let P be a program. Then
(1) the least model state, M_P^N, of P equals the least fixpoint of the operator ${}_NT_P$, ${}_NT_P\uparrow\omega$.

(2) $T_P\uparrow\omega \equiv_m {}_NT_P\uparrow\omega$.

Proof:

Part 1: $M_P^N = \bigcap\{K \mid K \text{ is a model state of P}\} = glb\{K \mid K \text{ is a model state of P}\}$
$= glb\{K \mid {}_NT_P(K) \subseteq K\} = lfp({}_NT_P) = {}_NT_P\uparrow\omega$.

Part 2: We prove (2) by showing that for $n \geq 0$, $T_P\uparrow n \equiv_m {}_NT_P\uparrow n$.

First, it is easy to prove that for $m \geq 0$, $T_P\uparrow m \subseteq {}_NT_P\uparrow m$: ${}_NT_P\uparrow 0 = T_P\uparrow 0$, and $T_P\uparrow k \subseteq {}_NT_P\uparrow k$ implies $T_P\uparrow(k+1) \subseteq T_P({}_NT_P\uparrow k) \subseteq equ({}_NT_P\uparrow k) \cup T_P({}_NT_P\uparrow k) = {}_NT_P\uparrow(k+1)$.

For $n=0$, ${}_NT_P\uparrow 0 = T_P\uparrow 0$, so $T_P\uparrow 0 \equiv_m {}_NT_P\uparrow 0$.

Assume that for $n<k$, $T_P\uparrow n \equiv_m {}_NT_P\uparrow n$.

In the appendix we show that for any subset I of HBNV(P) and any $n>0$, if $T_P\uparrow n \equiv_m I$ and $T_P\uparrow n \subseteq I$, then $T_P\uparrow(n+1) \equiv_m T_P(I)$. Since $T_P\uparrow(k-1) \subseteq {}_NT_P\uparrow(k-1)$ and by the induction hypothesis, $T_P\uparrow(k-1) \equiv_m {}_NT_P\uparrow(k-1)$, we have $T_P\uparrow k \equiv_m T_P({}_NT_P\uparrow(k-1))$. In any S-mapping V, $({}_NT_P\uparrow k)_V = ({}_NT_P\uparrow(k-1))_V \cup (T_P({}_NT_P\uparrow(k-1))_V$. Since ${}_NT_P\uparrow(k-1) \equiv_m T_P\uparrow(k-1)$ and $T_P({}_NT_P\uparrow(k-1)) \equiv_m T_P\uparrow k$, we have $({}_NT_P\uparrow k)_V = (T_P\uparrow(k-1))_V \cup (T_P\uparrow k)_V$. And since T_P is continuous, $T_P\uparrow(k-1) \subseteq T_P\uparrow k$. So, $({}_NT_P\uparrow k)_V = (T_P\uparrow k)_V$, and hence ${}_NT_P\uparrow k \equiv_m T_P\uparrow k$. *Q.E.D.*

Since the most useful information contained in a least model state of a program is the "images" of that model state in each possible world, (2) above tells us that it is adequate to calculate the least fixpoint of the T_P operator.

4. Procedural Semantics

In the previous sections, we have defined declarative semantics for programs with null values. This includes both a model theoretic and a fixpoint characterization. In this section, we present a refutation procedure, called NSLD refutation, which is a generalization of SLD refutation with a discussion of how it relates to the declarative semantics.

4.1. Resolution

Our procedural semantics is based on the concept of resolution. The following definitions are adapted from the binary resolution for clauses in [3].

Definition D4.1 (*Factor*) Let $C = (C;w)$ be an extended clause. If two or more literals (with the same sign) C have a most general unifier μ, and if $w[\mu]_A \neq \bot$, then $C\mu = (C\mu; w[\mu]_A)$ is called a *factor* of C. ■

Example 4.1

Let (C;w) be the extended clause $(P(\sigma_1,x) \vee P(z,f(y)) \vee Q(a,c);\{\sigma_1\} \subseteq \{a,b,c\})$. The first and second literals of C have an mgu $\mu = \{\sigma_1/z, f(y)/x\}$. Since $[\mu]_A = \emptyset$, $w[\mu]_A = w \neq \bot$. Hence, $(C\mu; w[\mu]_A) = (P(\sigma_1,f(y)) \vee Q(a,c);w)$ is a factor of (C;w). ■

Definition D4.2 (*Binary Resolvent*) Let P be a program. Let $C_1 = (C_1;u)$ and $C_2 = (C_2;v)$ be two extended clauses such that there are no common variables in C_1 and C_2. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have a most general unifier θ, such that $[\theta]_A uv \neq \bot$, then we say that C_1 and C_2 ***resolve on*** L_1 and L_2 with ***binary resolvent*** $C = (C;w)$, where $C = (C_1\theta - L_1\theta) \cup (C_2\theta - L_2\theta)$ and $w = [\theta]_A uv$ C_1 and C_2 are called the *parent clauses* of C. ■

Example 4.2

Consider the two extended clauses $C_1 = (P(\sigma_1,f(y)) \vee Q(a,c);u)$, where u is $\{\{\sigma_1\} \subseteq \{a,b,c\}\}$, and $C_2 = (R(\sigma_2,x) \vee \neg P(b,x);v)$, where v is $\{\{\sigma_2\} \subseteq \{d,e,f\}\}$. We see from example 2.10, that $P(\sigma_1,f(y))$ and P(b,x) unifies with mgu $\theta = \{b/\sigma_1, f(y)/x\}$, and $[\theta]_A uv = \{\{\sigma_1\} \subseteq \{b\}, \{\sigma_2\} \subseteq \{d,e,f\}\} \neq \bot$. Therefore, C_1 and C_2 can resolve on $P(\sigma_1,f(y))$ with the binary resolvent

$C = (Q(a,c) \vee R(\sigma_2,f(y)); \{\{\sigma_1\} \subseteq \{b\}, \{\sigma_2\} \subseteq \{d,e,f\}\})$

■

Definition D4.3 (*Resolvent*) A ***resolvent*** of (parent) extended clauses $(C_1;w_1)$ and $(C_2;w_2)$, is one of the following binary resolvents:

1. a binary resolvent of $(C_1;w_1)$ and $(C_2;w_2)$,
2. a binary resolvent of $(C_1;w_1)$ and a factor of $(C_2;w_2)$,

3. a binary resolvent of a factor of $(C_1;w_1)$ and $(C_2;w_2)$,
4. a binary resolvent of a factor of $(C_1;w_1)$ and a factor of $(C_2;w_2)$, ■

Example 4.3
Let $C_1 = (C_1;w_1)$ be the extended clause $(P(\sigma_1,x) \vee P(z,f(y)) \vee Q(a,c);\{\sigma_1\} \subseteq \{a,b,c\})$, and $C_2 = (C_2;w_2)$ be the extended clause $(R(\sigma_2,x) \vee \neg P(b,x);\{\sigma_2\} \subseteq \{d,e,f\})$. From example 4.1, we know that a factor of C_1 is $C_1' = (P(\sigma_1,f(y)) \vee Q(a,c);\{\sigma_1\} \subseteq \{a,b,c\})$, and from the previous example, we know that a binary resolvent of C_1' and C_2 is $C = (Q(a,c) \vee R(\sigma_2,f(y));\{\{\sigma_1\} \subseteq \{b\},\{\sigma_2\} \subseteq \{d,e,f\}\})$. Therefore, C is a resolvent of C_1 and C_2. ■

The resolution theorem below states that a resolvent is a logical consequence of its parent clauses. In proving it, we need the following lemma.

Lemma L4.1 Let P be a program, and $C = (C;w)$ be an extended clause. Let $P \models_N C$.
(1) if $C\mu$ is a factor of C, then $P \models_N C\mu$.
(2) If u is an assignment such that $uw \neq \bot$ then $P \models_N (C;uw)$.
(3) If θ is a substitution such that $w[\theta]_A \neq \bot$, then $P \models_N (C\theta;w[\theta]_A)$.
Proof:
(1) Let V be an S-mapping such that $w[\mu]_A[\rho_V]_A \neq \bot$. Then $w[\rho_V]_A \neq \bot$. Since $P \models_N (C;w)$, this means that for any model state I of P, $I_V \models C\rho_V$. Since $C\mu$ is just an instance of C, $I_V \models C\mu\rho_V$. Hence, $P \models_N (C\mu;w[\mu]_A)$.
(2) Let V be an S-mapping such that $uw[\rho_V]_A \neq \bot$. Then $w[\rho_V]_A \neq \bot$. Since $P \models_N (C;w)$, this means that for any model state I of P, $I_V \models C\rho_V$. Hence, $P \models_N (C;uw)$.
(3) We need to prove that for any model state, I, of P, and all S-mapping V such that $(w[\theta]_A)[\rho_V]_A \neq \bot$, I_V is a Herbrand model of $C\theta\rho_V$.
If $(w[\theta]_A)[\rho_V]_A \neq \bot$ then $w[\rho_V]_A \neq \bot$ and since $P \models_N (C;w)$, I_V is a Herbrand model of $C\rho_V$. Therefore, I_V is a Herbrand model of all ground instances of $C\rho_V$. To prove that I_V is a model of $C\theta\rho_V$, we need to prove that I_V is a model of all the ground instances of $C\theta\rho_V$. We can show this by proving that any ground instance of $C\theta\rho_V$ is a ground instance of $C\rho_V$.
Let $\theta = \theta_v \cup \theta_s$, where θ_v contains variable substitutions only, and θ_s contains S-constant substitutions only. $(w[\theta]_A)[\rho_V]_A \neq \bot$ implies that $[\theta]_A[\rho_V]_A \neq \bot$, and so $\theta_s\rho_V = \rho_V$ (see appendix) and therefore, $\theta\rho_V = \theta_v\rho_V$ and since θ_v contains only variable substitutions and ρ_V contains only S-constant substitutions, $\theta_v\rho_V = \rho_V(\theta_v\rho_V)$. Hence, for any variable substitution γ, $(C\theta\rho_V)\gamma = (C\rho_V)\theta_v\rho_V\gamma$, and any ground instance of $C\theta\rho_V$ is a ground instance of $C\rho_V$. *Q.E.D.*

Theorem T4.1 (***Resolution Theorem***) Let P be a program, and C_1, C_2 be two extended clauses. If C is a resolvent of C_1 and C_2, and if $P \models_N C_1$ and $P \models_N C_2$, then $P \models_N C$.
Proof:
From lemma L4.1(1), we know that if $C_1\mu_1$ and $C_2\mu_2$ are factors of C_1 and C_2, respectively, then
(T4.1.1) $P \models_N C_1\mu$ and $P \models_N C_2\mu_2$
Therefore, in the following, we only show the proof for case (1) in the definition of resolvent, i.e. the resolvent being a binary resolvent of C_1 and C_2. The cases involving factor(s) of C_1 or C_2 or both, are straight forward implication of (T4.1.1) and case (1).
Let C_1 be $(C_1;u) = (L_1 \vee L_{11} \vee \ldots \vee L_{1m};u)$ and C_2 be $(C_2;v) = (L_2 \vee L_{21} \vee \ldots \vee L_{2n};v)$ such that C_1 and C_2 resolve on L_1 and L_2 with mgu θ and resolvent $C = (C;w)$. Then
$C = (L_{11} \vee \ldots \vee L_{1m} \vee L_{21} \vee \ldots \vee L_{2n})\theta$ and $w = [\theta]_A uv \neq \bot$.
$w = [\theta]_A uv \neq \bot$ implies that $[\theta]_A u \neq \bot$ and $[\theta]_A v \neq \bot$, and since $P \models_N C_1$ and $P \models_N C_2$, from lemma L4.1(3), $P \models_N (C_1\theta;u[\theta]_A)$ and $P \models_N (C_2\theta;v[\theta]_A)$. Also, $w = [\theta]_A uv \neq \bot$, implies that, for any S-mapping V, if $[\rho_V]_A w \neq \bot$ then $[\rho_V]_A u[\theta]_A \neq \bot$, and $[\rho_V]_A v[\theta]_A \neq \bot$. Therefore, for any model state M of P, $M_V \models C_1\theta\rho_V$ and $M_V \models C_2\theta\rho_V$. We need to prove that $M_V \models C\rho_V$.
Assume that $C\rho_V$ is false w.r.t. M_V. Then $L_{1i}\theta\rho_V$ and $L_{2j}\theta\rho_V$ are all false w.r.t. M_V, for $1 \le i \le m$ and $1 \le j \le n$. In order for $M_V \models C_1\theta\rho_V$ and $M_V \models C_2\theta\rho_V$, we must have $M_V \models L_1\theta\rho_V$ and $M_V \models L_2\theta\rho_V$. Since θ is an mgu of L_1 and $\neg L_2$, $L_1\theta = \neg L_2\theta$. Therefore, both $L_1\theta\rho_V$ and $L_2\theta\rho_V = \neg L_1\theta\rho_V$ are true

w.r.t. M_V, a contradiction. Hence, $M_V \models C$, and M is a model state of C. *Q.E.D.*

4.2. NSLD-Refutation

NSLD-refutation is an extension of SLD-refutation to logic programs with null values.

Definition D4.4 (***NSLD-Derivation***) Let P be a program and $G=(\leftarrow A_1,...,A_k;w)$ be an extended goal clause. An ***NSLD-derivation*** of $P \cup \{G\}$ is a sequence of extended goal clauses G_0, G_1 ... where $G_0=G$ and for all $i\geq 0$, G_{i+1} is obtained from G_i as follow:

(1) G_i is $(\leftarrow A_{i1},...,A_{im},...,A_{ik};w_i)$
(2) $C_{i+1}=(A\leftarrow B_1,...,B_q;u)$ is an extended program clause in P
(3) $A_{im}\theta_{i+1}=A\theta_{i+1}$, where θ_{i+1} is an mgu of A and A_{im}.
(4) $w_{i+1}=w_i[\theta_{i+1}]_A\neq\perp$
(5) G_{i+1} is $(\leftarrow(A_{i1},...,A_{im-1},B_1,...,B_q,A_{im+1},...,A_{ik})\theta_{i+1};w_{i+1})$

We call C_{i+1} the *input clause* to the i+1-st step. ■
Note that by (4) above, a sequence of extended goal clauses is an NSLD derivation only if the assignment part of each clause is not $\perp$.

Definition D4.5 (***NSLD-Refutation***) Let P be a program, and G=(g;u) be an extended goal clause. An ***NSLD-refutation*** of $P \cup \{G\}$ is a finite NSLD-derivation of $P \cup \{G\}$ such that $G_n=(\square;u[\theta_1]_A...[\theta_n]_A)$, where θ_i, $1\leq i\leq n$, are the mgu's in the derivation. We say the refutation has length n. ■

Example 4.4
Consider the program in Example 2.11:

(1) $(F(\sigma_1,e);\{\{\sigma_1\}\subseteq\{e,f,g\}\})$
(2) $(F(\sigma_2,f);\{\{\sigma_2\}\subseteq\{f,g,h\}\})$
(3) $(S(x,y)\leftarrow F(z,x),F(z,y);\varnothing)$

Let G be the goal $(\leftarrow S(e,f);\varnothing)$. One possible NSLD-refutation is as follows:
Step 1: G_0 is $(\leftarrow S(e,f);\varnothing)$. We choose S(e,f) and use clause (3) to calculate G_1. The mgu for the two atoms is $\theta_1 = \{e/x,f/y\}$. Since there is no S-constant in θ_1, $w_1=[\theta_1]_A=\varnothing$. And G_1 is $(\leftarrow F(z,e),F(z,f);\varnothing)$.
Step 2: Choose F(z,e) and use clause (1). The mgu is $\theta_2 = \{\sigma_1/z\}$. This does not involve substitution of S-constants, therefore, w_2 is still $\varnothing$. And G_2 is $(\leftarrow F(\sigma_1,f);\varnothing)$.
Step 3: The only atom left is $F(\sigma_1,f)$. We use clause (2). The mgu is $\theta_3 = \{\sigma_2/\sigma_1\}$. $w_3=w_2[\theta_3]_A$ which is $\{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\}$. And G_3 is the extended null clause $(\square;w_3)$.
This is, therefore, an NSLD-refutation of length 3. ■

4.3. Soundness of NSLD-Resolution

We can (informally) demonstrate the soundness of NSLD-refutation as a direct result of the resolution theorem. In the definition of NSLD-derivation, we see that if G_{i+1} is obtained from G_i by using an extended clause $C = (A\leftarrow B_1,...,B_q;u)$ and an mgu θ_{i+1}, then C and G_i resolve on A with mgu θ_{i+1} and resolvent G_{i+1}. Since C is a clause in P, $P\models_N C$. Therefore, by the resolution theorem, if $P\models_N G_i$ then $P\models_N G_{i+1}$. Now let $G_0=(\leftarrow A_1,...,A_n;w)$. By induction on the length of an NSLD-refutation for $P \cup \{G_0\}$, we will have $P\models_N G_0$ implies $P\models_N(\square;w)$. To put it another way, if we assume $\neg(A_i\wedge...\wedge A_n)$, $1\leq i\leq n$, is true in P, then we will derive the empty clause, i.e. a contradiction. Therefore, A_i must all be true. Of course, to formally prove this soundness result, we need to consider the substitutions obtained from the refutation. The following definitions are useful in subsequent discussions.

Definition D4.6 (***Computed Answer***) Let P be a program, G=(g;u) be an extended goal. If there is an n-length NSLD refutation of $P \cup \{G\}$ with mgu's $\theta_1,...,\theta_n$, then a ***computed answer substitution*** θ for $P \cup \{G\}$ is the substitution obtained by restricting the variables in the composition $\theta_1...\theta_n$ to the variables that appear in g, and a *computed answer assignment* is the assignment $[\theta_1]_A...[\theta_n]_A$. ■
In example 4.4 the computed answer substitution is ε, and computer answer assignment is $[\theta_1]_A[\theta_2]_A[\theta_3]_A = \varnothing\varnothing[\theta_3]_A = [\theta_3]_A = \{\{\sigma_1,\sigma_2\}\subseteq\{f,g\}\}$.

Now we formally state a result regarding the soundness of NSLD-refutation with respect to the declarative semantics presented in the previous sections. The following lemmas are used in the proof of the soundness theorem.

Lemma L4.2 Let P be a program, and $C = (C;w)$ be an extended clause. Let $P \models_N C$.
(1) If u is an assignment such that $uw \neq \bot$ then $P \models_N (C;uw)$.
(2) If θ is a substitution such that $w[\theta]_A \neq \bot$, then $P \models_N (C\theta;w[\theta]_A)$.
Proof:
Let V be an S-mapping
(1) $uw[\rho_V]_A \neq \bot$
$\Rightarrow w[\rho_V]_A \neq \bot$
$\Rightarrow P_V \models C\rho_V$ ($\because P_V \models_N (C;w)$)
$\Rightarrow P \models_N (C;uw)$.
(2) $w[\theta]_A[\rho_V]_A \neq \bot$
$\Rightarrow w[\rho_V]_A \neq \bot$
$\Rightarrow P_V \models C\rho_V \Rightarrow P_V \models C\theta\rho_V$
($\because \rho_V$ only involves substituting S-constants for constants.)
$\Rightarrow P \models_N (C\theta;w[\theta]_A)$. *Q.E.D.*

Lemma L4.3 Let P be a program. Then
(1) (a) If $P \models_N (A \wedge B ; w)$ then $P \models_N (A ; w)$ and $P \models_N (B ; w)$
(b) If $P \models_N (A ; u)$ and $P \models_N (B ; w)$ and $uw \neq \bot$ then $P \models_N (A \wedge B ; uw)$
(2) $P \models_N (A \leftarrow B ; w)$ and $P \models_N (B ; w)$ implies $P \models_N (A ; w)$
Proof:
(1a) $P \models_N (A \wedge B;w)$
$\Rightarrow$ for any model state I of P, and for any S-mapping V such that $w\rho_V \neq \bot$, $I_V \models (A \wedge B)\rho_V$
$\Rightarrow$ for any model state I of P, and for any S-mapping V such that $w\rho_V \neq \bot$, $I_V \models A\rho_V$ and $I_V \models B\rho_V$
$\Rightarrow P \models_N (A;w)$ and $P \models_N (B;w)$.
(1b) $uw \neq \bot$, and $P \models_N (A;u)$, $P \models_N (B;w)$
$\Rightarrow$ for any S-mapping V, if $uw\rho_V \neq \bot$ then $u\rho_V \neq \bot$ and $w\rho_V \neq \bot$. And for any model state I of P, and any S-mapping V′ such that $u\rho_{V'} \neq \bot$ and $w\rho_{V'} \neq \bot$, $I_{V'} \models A\rho_{V'}$ and $I_{V'} \models B\rho_{V'}$
$\Rightarrow$ for any model state I of P and for any S-mapping V such that $uv\rho_V \neq \bot$, $I_V \models A\rho_V \wedge B\rho_V$
$\Rightarrow P \models_N (A \wedge B;uv)$.
(2) $P \models_N (A \leftarrow B;w)$ and $P \models_N (B;w)$
iff for any model state I of P, and for any S-mapping V such that $w\rho_V \neq \bot$, $I_V \models (A \leftarrow B)\rho_V$ and $I_V \models B\rho_V$
implies for any model state I of P, and for any S-mapping V such that $w\rho_V \neq \bot$, $I_V \models A\rho_V$
iff $P \models_N (A;w)$.
Q.E.D.

Theorem T4.2 (***Soundness***) Let P be a definite program, and $G=(\leftarrow C;u)$ be an extended goal clause. If there is an NSLD-refutation for $P \cup \{G\}$ with computed answer substitution Θ and computed answer assignment w, then $P \models_N (C\Theta;uw)$.
Proof:
Let $G=(\leftarrow C;u)=(\leftarrow A_1,...,A_m;u)$. Let the $P \cup \{G\}$ has an NSLD refutation with length n, and mgu's $\theta_1,...,\theta_n$. Then $w=[\theta_1]_A...[\theta_n]_A$. We prove the theorem by induction on the length n.
n=1: $C=A_1$, $\Theta=\theta_1$, $w=[\theta_1]_A$ and there is (A;v) in P such that $A\theta_1=A_1\theta_1$ and $u[\theta]_A \neq \bot$. Since (A;v) is an extended program clause in P, $P \models_N (A;\varnothing)$. By lemma L4.2(2), $P \models_N (A\theta_1;\varnothing[\theta_1]_A)$. Since $u[\theta_1]_A \neq \bot$, by lemma L4.2(1), $P \models_N (A\theta_1;u[\theta_1]_A) = (A_1\theta_1;uw) = (C\Theta;uw)$
n<k: Assume that the theorem is true for n<k.
n=k: Let the first input clause be $(A \leftarrow B_1,...,B_p;v)$ in P and the atom chosen from G_0 be A_i and θ_1 is such that $A\theta_1=A_i\theta_1$ and $u[\theta_1]_A \neq \bot$. Then G_1 is
$(\leftarrow C_1;w_1) = (\leftarrow (A_1,...,A_{i-1},B_1,...,B_p,A_{i+1},...,A_m)\theta_1;u[\theta_1]_A)$.

$G_1,...,G_n$ is a length $n-1$ NSLD-refutation for $P \cup \{G_1\}$, with mgu's $\theta_2,...,\theta_n$. By the induction hypothesis, $P \models_N (C_1\theta_2...\theta_n; w_1[\theta_2]_A...[\theta_n]_A)$.

By lemma L4.3(1), $P \models_N ((B_1,...,B_p)\theta_1\theta_2...\theta_n; w_1[\theta_2]_A...[\theta_n]_A) = ((B_1,...,B_p)\Theta; uw)$. Since $(A \leftarrow B_1,...,B_p; v)$ is an extended clause in P, $P \models_N (A \leftarrow B_1,...,B_p; \emptyset)$. By lemma L4.2(1), $P \models_N (A \leftarrow B_1,...,B_p; u)$. Since $uw = u[\theta_1]_A...[\theta_n]_A \neq \perp$, by lemma L4.2(1), $P \models_N ((A \leftarrow B_1,...,B_p)\Theta; uw)$. So by lemma L4.3(2), $P \models_N (A\Theta; uw)$. Also, by lemma L4.3(1), $P \models_N ((A_1,...,A_{i-1},A_{i+1},...,A_m)\Theta; uw)$. So, $P \models_N ((A_1,...,A_m)\Theta; uw)$. *Q.E.D.*

Example 4.5

Consider the refutation for the goal clause $(\leftarrow S(e,f); \emptyset)$ in Example 4.4. The computed answer substitution is $\theta = \theta_1\theta_2\theta_3 = \{e/x, f/y, \sigma_2/z, \sigma_2/\sigma_1\}$ with the variables restricted to those in the goal. Since there is no variable in the goal clause, the resulting substitution is just $\theta_3 = \{\sigma_2/\sigma_1\}$.

Now $(g\theta; w[\theta_1]_A[\theta_2]_A[\theta_3]_A) = (S(e,f); \{\{\sigma_1,\sigma_2\} \subseteq \{f,g\}\})$, which is an element in $T_P\uparrow\omega$ (see Example 3.2). Therefore, θ is a correct answer substitution. ■

4.4. Completeness of NSLD-Resolution

In this section we prove some completeness results of NSLD refutation. The main technique used in the proofs is combining refutations to form a larger one. However, the presence of S-constants complicates the proof in the following way. Without S-constants, the (SLD) refutation of a ground goal will always yield an answer substitution that is the identity substitution ε, and the refutations of two ground atoms can be easily combined to form a refutation for the conjunction of the two atoms. However, if there are S-constants in a ground goal, an NSLD refutation may produce an answer substitution that contains terms of the form c/σ or σ_1/σ_2, where c is a constant and σ,σ_1,σ_2 are S-constants. Therefore, we need the following lemma.

Lemma L4.4 Let P be a program with null values. Let $(B_i; w)$, $1 \le i \le n$, be extended atoms in HBNV(P). If $P \cup \{(\leftarrow B_i; w)\}$ has an NSLD-refutation with computed answer substitution ε and computed answer assignment w, for $1 \le i \le n$, then $P \cup \{(\leftarrow B_1,...,B_n; w)\}$ also has an NSLD-refutation with computed answer substitution ε and computed answer assignment w.

Proof:

$n=1$: trivial.

$n>1$: Let G_0 be $(\leftarrow B_1,...,B_n; w)$. Let the NSLD-refutation for $P \cup \{(\leftarrow B_i; w)\}$ consists of the sequence of goals $(G_{i0},...,G_{im_i})$, the sequence of input clauses $(C_{i1},...,C_{im_i})$, and the sequence of mgu's $(\theta_{i1},...,\theta_{im_i})$, where $G_{ij} = (\leftarrow g_{ij}; w_{ij})$, for $0 \le j \le m_i$, and $g_{i0} = B_i$, and $g_{im_i} = \square$. We can construct a sequence of goals $(G_1,...,G_m)$ such that $G_j = (\leftarrow g_{ij}, b_j; ww_{ij})$, where b_j is the (ground) formula $(B_1,...,B_{i-1},B_{i+1},...,B_n)\theta_{i1}...\theta_{ij}$. Since $w = w_{i1}...w_{im_i}$, and since each of the B_k, $1 \le k \le n$, is ground, $B_k\theta_{i1}...\theta_{im_i} = B_k[w_{i1}]_A...[w_{im_i}]_A = B_k[w]_S$. So, G_m would be the goal $(\leftarrow (B_1,...,B_n)[w]_S; w)$. For each B_k, $1 \le k \le n$, $(B_k; w)$ is in HBNV(P), hence $B_k[w]_S = B_k$. Therefore, $G_m = (\leftarrow B_1,...,B_{i-1},B_{i+1},...,B_n; w)$. Since there are only $n-1$ atoms in G_m, we can use the induction hypothesis. Hence, G_0 has an NSLD-refutation. *Q.E.D.*

The following property of the fixpoint operator T_P makes the proof of completeness much easier.

Proposition P4.1 Let P be a program with null values. Then $T_P\uparrow n$ can be reformulated as $T_P\uparrow 1 = T_P(\emptyset)$ $T_P\uparrow n = \{(A; w) \mid (A \leftarrow B_1, \ldots, B_n; w) \in Gnd(P)$, and there are $(B_1; w),...,(B_n; w)$ in $T_P\uparrow(n-1)\}$, $k>1$.

Proof:

By the definition of T_P, $(A; w) \in T_P\uparrow n$ iff there is $(A \leftarrow B_1,...,B_n)$ in Gnd(P) and there are $(B_1'; w_1)$,..., $(B_n'; w_n)$ in $T_P\uparrow(n-1)$ such that $w = w_1...w_n$ and $B_i'[w]_S = B_i$ for $1 \le i \le n$.

It is easy to prove by induction that $T_P\uparrow k$ is closed under ground instance, i.e. if $(B; v) \in T_P\uparrow k$ for some k then any ground instance of $(B; v)$ is also in $T_P\uparrow k$.

Since $B_i'[w]_S$ is ground, and from the appendix we know that $[[w]_S]_A = w$, $(B_i'[w]_S; w_i w) = (B_i; w)$ is a ground instance of $(B_i'; w_i)$. Therefore, $(B_i; w) \in T_P\uparrow(n-1)$. *Q.E.D.*

Note that the above reformulation only applies to the upward powers of T_P, and not to the definition of T_P itself.

Theorem T4.3 (***Completeness***) Let P be a program with null values. Let (A;w) be an element in $T_P\uparrow\omega$, then $P \cup \{(\leftarrow A;w)\}$ has an NSLD-refutation, such that the computed answer substitution is ε and the computed answer assignment is w.

Proof:

Let (A;w) be an element in $T_P\uparrow n$. We prove by induction on n.

n=1: $(A;w) \in T_P\uparrow 1$. $(A\leftarrow;w) \in Gnd(P)$. That is, there is a unit clause $(A'\leftarrow;u)$ in P and a pair of substitutions θ_v and θ_s, where θ_v contains only variable substitutions and θ_s contains only S-constant substitutions, such that $A'\theta_v\theta_s=A$ and $[\theta_s]_A=w$.
Since $A'\theta_v\theta_s=A'\theta_s(\theta_v\theta_s)=A$, and $[w]_S=\theta_s$, therefore, there is an NSLD-refutation of length 1 where the unifier is $\theta_v\theta_s$.

n>1: Let (A;w) be an element in $T_P\uparrow n$. Then there is $(A\leftarrow B_1,...,B_m;w) \in Gnd(P)$, such that for $1\le i\le m$, $(B_i;w) \in T_P\uparrow(n-1)$. By the definition of Gnd(P), there exist an extended clause $C=(A'\leftarrow B'_1,...,B'_m;u)$ in P, a variable substitution θ_v, and an S-constant substitution θ_s such that $[\theta_s]_A=w$ and $A'\theta_v\theta_s=A$, and $B'_i\theta_v\theta_s=B_i$ for $1\le i\le m$.
$A'\theta_v\theta_s=A'\theta_s(\theta_v\theta_s)=A$ and $B'\theta_v\theta_s=B'\theta_s(\theta_v\theta_s)=B_i$ for $1\le i\le m$. Therefore, A and the head of the extended clause $C_w=(A'[w]_S\leftarrow B'_1[w]_S,...,B'_m[w]_S;uw)=(A'\theta_s\leftarrow B'_1\theta_s,...,B'_m\theta_s;w)$ unifies with mgu $\theta_v\theta_s$. If we let $G_0=(\leftarrow A;w)$, and use C as the input clause, then we have $G_1=(\leftarrow(B'_1\theta_s,...,B'_m\theta_s)(\theta_v\theta_s);[\theta_s]_A)=(\leftarrow B_1,...,B_m;w)$. Since $(B_i;w) \in T_P\uparrow(n-1)$, by the induction hypothesis, each $(B_i;w)$ has an NSLD-refutation with computed answer substitution ε and computed answer assignment w. Using lemma L4.4, we have our result. *Q.E.D.*

Furthermore, from theorem T3.1(2), we have the following (weak) completeness regarding ${}_N T_P$.

Theorem T4.4 (***Weak Completeness***) Let P be a program with null values. Let (A;w) be an element in M_P^N, then there are m NSLD-refutation, $m\ge 1$, for $P \cup \{(\leftarrow A;w)\}$ with computed answer substitutions $\Theta_1,...,\Theta_m$, and computed answer assignments $u_1,...,u_m$, such that $\{(A;w)\} \equiv_m \bigcup_{1\le i\le m} \{(A\Theta_i;wu_i)\}$. ■

This result states that NSLD can calculate the images for every element in M_P^N, but it might not be able to calculate the elements directly. For example, if a program contains the following clauses: $(P(x)\leftarrow Q(x,y),R(y);\varnothing)$, $(Q(a,\sigma);\{\sigma\} \subseteq \{b,c\}\})$, $(R(b);\varnothing)$ and $(R(c);\varnothing)$. Then $(P(a);\varnothing)$ is in M_P^N. However, $(\leftarrow P(a);\varnothing)$ only have two NSLD refutations, with computed answer assignments $w_1=\{\{\sigma\} \subseteq \{b\}\}$ and $w_2=\{\{\sigma\} \subseteq \{c\}\}$, respectively. It is easy to see that $\{(P(a);\varnothing)\} \equiv_m \{(P(a);w_1),(P(a);w_2)\}$.

5. Conclusion

In this paper, we consider the problems of representing, in definite logic programs, a class of indefinite information called null values, and of computing answers to queries in such programs.

To achieve this, we add to first order logic a new type of objects called S-constants. Together with other extensions, these S-constants are used to formulate the incompleteness represented by null values.

We then defined an extension to Herbrand models that gives our logic a model-theoretic semantics. Two operators are then defined to give a fixpoint characterization of this same semantics. One of the operator has a least fixpoint that corresponds exactly to the model semantics. The other operator captures the essential information contained in the model semantics and is much easier to compute than the first one. A proof procedure that is very similar in form to SLD resolution is also developed, with the proper soundness and completeness results.

Future work is to develop an appropriate semantics and proof procedure for inferring negative information from a program that contains null values. We then plan to investigate how to extend these work on null values to general Horn programs, i.e. programs that allow negative literals in the rule bodies.

Acknowledgement

We are grateful to Dr. John Grant, Dr. Arcot Rajasekar and Jorge Lobo for their helpful suggestions.
This work is done under the financial support of National Science Foundation (grant no. IRI-86-09170), the Army Research Office (grant no. DAAL-03-88-K0087), and the Air Force Office of Scientific Research (grant no. AFOSR-88-0152).

Appendix A.

In the proof of the following proposition, we use the following property of a directed set: ([12])
If **X** is a directed subset of a complete lattice, and if $\{A_1,...,A_n\} \subseteq lub(\mathbf{X})$ then $\{A_1,...,A_n\} \subseteq I$ for some $I \in \mathbf{X}$. (Assume that $A_i \in I_i$ for some I_i in **X**, then since **X** is directed, there is a set I in **X** such that $\bigcup_{1\le i\le n} I_i \subseteq I$, and hence $\{A_1,...,A_n\} \subseteq I$.)

Theorem TA.1 (*Continuity of* T_P and ${}_NT_P$) Given a program **P**, the T_P and the ${}_NT_P$ operators are continuous.

Proof:

We first prove the continuity of T_P. Let **X** be a directed subset of the power set of HBNV(P).

$(A;w) \in T_P(lub(\mathbf{X}))$
iff $(A \leftarrow B_1,...,B_n;w_0)$ is a ground instance of an extended clause in **P** and $(B_i;w_i) \in lub(\mathbf{X})$, for $1\le i\le n$, and $w=w_0w_1...w_n\neq\perp$
iff there is an I in **X** such that $(B_i;w_i) \in I$, for $1\le i\le n$.
iff $(A;w) \in T_P(I)$ for some $I \in \mathbf{X}$
iff $(A;w) \in \bigcup_{I\in\mathbf{X}} T_P(I) = lub(T_P(\mathbf{X}))$.

Next, since **X** is directed, any finite subset $\{I_1,...,I_n\}$ of **X** has an upper bound J in **X**, i.e. $\bigcup_{1\le i\le n} I_i \subseteq J$.

Therefore,

$$equ(\bigcup_{1\le i\le n} I_i) \subseteq equ(J) \subseteq \bigcup\{equ(J) \mid J \in \mathbf{X}\} = lub(equ(\mathbf{X})),$$

and

$$equ(lub(\mathbf{X})) \subseteq lub(equ(\mathbf{X})).$$

Also, for any I and J, $equ(I) \cup equ(J) \subseteq equ(I \cup J)$, hence $lub(equ(\mathbf{X})) \subseteq equ(lub(\mathbf{X}))$. So, equ is continuous, and so is ${}_NT_P$. *Q.E.D.*

Proposition PA.1 (*Properties of* T_P) Let **P** be a program
(1) For any S-mapping V, an atom A is in $(T_P\uparrow n)_V$ iff there is an extended atom $(A;w)$ in $T_P\uparrow n$ such that $w[\rho_V]_A\neq\perp$.
(2) If I is a subset of HBNV(P) such that $I \equiv_m T_P\uparrow n$ and $T_P\uparrow n \subseteq I$ then $T_P\uparrow(n+1) \equiv_m T_P(I)$.

Proof:

Part I: When n=1, if A is in $(T_P\uparrow 1)_V$ then there is $(A';w')$ in $T_P\uparrow 1$ such that $w'[\rho_V]_A\neq\perp$ and $A'\rho_V=A$. Therefore, there is a ground instance $(A';w_0)$ of a program clause C in **P**, such that $w'=w_0$. This means that $(A'\rho_V;w_0[\rho_V]_A)$ is also a ground instance of C, since $w_0[\rho_V]_A\neq\perp$. Therefore, $(A'\rho_V;w_p[\rho_V]_A)=(A;w) \in T_P\uparrow 1$.
Assume that (1) is true for $n<k$. We want to show that it also holds for $n\ge k$.
If $A \in (T_P\uparrow k)_V$, then there is $(A';w')$ in $T_P\uparrow n$ such that $w'[\rho_V]_A\neq\perp$ and $A'\rho_V=A$. By definition of $T_P\uparrow n$, there is a program clause C in **P** that has a ground instance $(A' \leftarrow B_1,...,B_n;w_0)$ such that there are $(B_i;w_i) \in T_P\uparrow(k-1)$, $1\le i\le n$, and $w'=w_0w_1...w_n\neq\perp$. Since $w'[\rho_V]_A\neq\perp$, $w_i[\rho_V]_A\neq\perp$ for $0\le i\le n$. Therefore, $B_i\rho_V \in (T_P(k-1))_V$, $1\le i\le n$, and $(A'\rho_V \leftarrow B_1\rho_V,...,B_n\rho_V;w_0[\rho_V]_A)$ is also a ground instance of C. From the induction hypothesis, there are $(B\rho_V;u_i) \in T_P\uparrow(k-1)$, $1\le i\le n$, such that $u_i[\rho_V]_A\neq\perp$. Therefore, $w=w_0[\rho_V]_Au_1...u_n\neq\perp$, and $(A;w)=(A'\rho_V;w) \in T_P\uparrow k$.

Part II: We need to prove that for any S-mapping V, $(T_P\uparrow(n+1))_V=(T_P(I))_V$.
Since T_P is continuous, $T_P\uparrow(n+1) \subseteq T_P(I)$, and hence $(T_P\uparrow(n+1))_V \subseteq (T_P(I))_V$.
If $A \in (T_P(I))_V$ then there is $(A';w')$ in $T_P(I)$ such that $w'[\rho_V]_A\neq\perp$ and $A'\rho_V=A$. Therefore, there is a clause C in **P** that has a ground instance $(A' \leftarrow B_1,...,B_n;w_0)$ and $(B_i;w_i) \in I$, $1\le i\le n$, such that $w'=w_0w_1...w_n\neq\perp$. Since $w'[\rho_V]_A\neq\perp$, $w_i[\rho_V]_A\neq\perp$, $0\le i\le n$. So $B_i\rho_V \in I_V$. Since $T_P\uparrow n \equiv_m I$, $B_i\rho_V \in (T_P\uparrow n)_V$. By (1), there are $(B_i\rho_V;w_i) \in T_P\uparrow n$, $1\le i\le n$, such that $w_i[\rho_V]_A\neq\perp$. Since $w_0[\rho_V]_A\neq\perp$, $(A'\rho_V \leftarrow B_1\rho_V,...,B_n\rho_V;w_0[\rho_V]_A)$ is also a ground instance of C. Also, $w=w_0[\rho_V]_Aw_1...w_n\neq\perp$. Hence $(A'\rho_V;w)=(A;w) \in T_P(n+1)$, and $(T_P(I))_V \subseteq (T_P(n+1))_V$. *Q.E.D.*

Proposition PA.2 (*Properties of Assignments and Substitutions*) Let P be a program, and let θ be an applicable substitution w.r.t. P. Let $\theta=\theta_v \cup \theta_s$, where θ_v is the part of θ that involves only variable substitutions, and θ_s is the part that contains only S-constant substitutions. Let V be an S-mapping such that $[\theta]_A[\rho_V]_A \neq \perp$. Then $\theta\rho_V=\theta_v\rho_V$.

Proof:

$[\theta]_A=[\theta_s]_A$, $\therefore$ $[\theta_s]_A[\rho_V]_A \neq \perp$. Assume that ρ_V is $\{c_1/\sigma_1,...,c_n/\sigma_n\}$ and that θ_s is $\{t_1/v_1,...,t_m/v_m\}$, where v_i is one of σ_j, $1\le i\le m$ and $1\le j\le n$. $\theta_s\rho_V$ is $\{t_1\rho_V/v_1,...,t_m\rho_V/v_m,c_1/\sigma_1,..,c_n/\sigma_n\}$ with any redundant terms removed. In order for $\theta_s\rho_V=\rho_V$, we need to prove that for $1\le i\le m$, either $t_i\rho_V=v_i$ so that $t_i\rho_V/v_i$ is removed, or $t_i\rho_V=c_j$, so that $t_i\rho_V/v_i$ and c_j/σ_j are identical. Since ρ_V only substitutes S-constants by constants, the first situation does not occur. We prove the second one by contradiction.

Assume that there is an i, $1\le i\le m$, such that $t_i\rho_V \neq c_j$. There are two cases to consider.

Case 1: t_i is some constant c. $t_i\rho_V=c$. If $c\neq c_j$, then $ran(\sigma_j,[\theta_s]_A) \cap ran(\sigma_j,[\rho_V]_A)=\emptyset$, $\therefore$ $[\theta_s]_A[\rho_V]_A=\perp$.

Case 2: t_i is some S-constant σ_p such that $j<p$. $t_i\rho_V=c_p$. If $c_p\neq c_j$, then, because $t_i/v_i=\sigma_p/\sigma_j \in \theta_s$, $Cla(\sigma_j,[\theta_s]_A)=Cla(\sigma_p,[\theta_s]_A)$ and $ran(\sigma_j,[\theta_s]_A)=ran(\sigma_p,[\theta_s]_A)$. On the other hand, $ran(\sigma_p,[\rho_V]_A) \cap ran(\sigma_j,[\rho_V]_A) = \{c_p\} \cap \{c_j\}=\emptyset$. These facts implies that $[\theta_s]_A[\rho_V]_A=\perp$. *Q.E.D.*

REFERENCES

1. BISKUP, J. A Formal Approach to Null Values in Database Relations, *Advances in Data Base Theory, vol. 1*, H. Gallaire, J. Minker, and J. M. Nicolas (eds.), Plenum Press, New York, 1981 , pp. 299-341.
2. BISKUP, J. A Foundation of Codd's Relational Maybe-operations, *XP2 Work-shop on Relational Database Theory* (University Park, June). Pennsylvania State Univ., 1981.
3. CHANG, C. L. AND R. C. T. LEE *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
4. CODD, E. F. Understanding Relations (Installment #7), *FDT Bull. of ACM-SIGMOD 7*, 3-4 (Dec. 1975) , pp. 23-28.
5. CODD, E. F. Extending the Database Relational Model to Capture More Meaning, *ACM Trans. Database Systems 4*, 4 (Dec. 1979) , pp. 397-434.
6. GRANT, J. Null Values in a Relational Data Base, *Information Processing Letters 6*, 5 (Oct. 1977) , pp. 156-157.
7. GRANT, J. AND J. MINKER Answering Queries in Indefinite Databases and the Null Value Problem, *Advances in Computing Research 3* (1986) , pp. 247-267.
8. IMIELINSKI, T. AND W. LIPSKI Incomplete Information in Relational Databases, *J. ACM 31*, 4 (Oct. 1984) , pp. 761-791.
9. IMIELINSKI, T. AND K. VADAPARTY Complexity of Query Processing in Databases with OR-Objects, *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Philadelphia, PA, March 29-31). ACM Press, 1989, pp. 51-65.
10. LASSEZ, J.L., M.J. MAHER, AND K. MARRIOTT Unification Revisited, *Foundations of Deductive Databases and Logic Programming*, J. Minker (ed.), Morgan Kaufmann, 1988 .
11. LIPSKI, W. On Semantic Issues Connected with Incomplete Information Databases, *ACM Trans. Database Systems 4*, 3 (Sept. 1979) , pp. 262-296.
12. LLOYD, J. W. *Foundations of Logic Programming*, Springer-Verlag, New York, 1984.
13. REITER, R. Towards a logical reconstruction of relational database theory, *Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming Languages*, M. L. Brodie, J. Mylopoulos, and J. Schmidt (eds.), Springer-Verlag New York, 1984 , pp. 191-233.
14. REITER, R. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values, *J. ACM 33*, 2 (April 1986) , pp. 349-370.
15. SIKLOSSY, L. Efficient Query Evaluation in Relational Databases with Missing Values, *Information Processing Letters 13*, 4-5 (1981) , pp. 160-163.
16. VARDI, M. Querying Logical Databases, *Proceedings of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Portland, OR, Mar. 25-27). ACM, New York, 1985, pp. 57-65.

17. YUAN, L. Y. AND D. A. CHIANG A Sound and Complete Query Evaluation Algorithm for Relational Databases with Null Values, *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Chicago, June). ACM, New York, 1988, pp. 74-81.
18. ZANIOLO, C. Database Relations with Null Values, *J. Comp. Sys. Sci. 28* (1984) , pp. 142-166.